Oracle Cloud Native Environment

Kubernetes Clusters for Release 2

F96197-24
March 2026

Oracle Cloud Native Environment Kubernetes Clusters for Release 2,

F96197-24

Contents

6 Oracle Linux Virtualization Manager Provider

7 OCI Provider

8 Bring Your Own Provider

Preface

This document includes information on using the Oracle Cloud Native Environment (Oracle CNE) Command Line Interface (CLI) to create and manage Kubernetes clusters using the available cluster providers.

1

Introduction

Learn about the high level options available to create Kubernetes clusters in Oracle Cloud Native Environment (Oracle CNE).

Oracle CNE includes a Command Line Interface (CLI) that can manage the life cycle of Kubernetes clusters, using OSTree based container images. The container images include both the host Oracle Linux OS, and the Kubernetes software distribution. These images are deployed to hosts or Virtual Machines (VMs) to create nodes in a Kubernetes cluster. These images are referred to in this documentation set as Oracle Container Host for Kubernetes (OCK) images.

Kubernetes clusters are created and managed using the CLI `ocne cluster` command. For the complete `ocne cluster` command options and syntax, see Oracle Cloud Native Environment: CLI.

Oracle CNE includes several provider types you can use to create a Kubernetes cluster. These providers use the OCK image to provision nodes in a cluster. Create clusters for:

- Kernel-based Virtual Machines (KVM) using the `libvirt` provider.
- Oracle Linux Virtualization Manager using the `olvm` provider.
- Oracle Cloud Infrastructure (OCI) using the `oci` provider.
- Custom installations for bare metal or other platforms using the `byo` provider.

The `libvirt` provider is the default cluster provider, and can be used to provision Kubernetes clusters using Kernel-based Virtual Machines (KVM). The default KVM stack includes libvirt, and is included, by default, with Oracle Linux.

Kubernetes clusters are deployed to Oracle Linux Virtualization Manager using the `olvm` provider. The `olvm` provider is an implementation of the Kubernetes Cluster API. For information about the Kubernetes Cluster API, see the upstream documentation. The `olvm` provider uses the oVirt REST API to communicate with Oracle Linux Virtualization Manager. For information on the oVirt REST API, see the oVirt REST API upstream documentation.

Kubernetes clusters are deployed to OCI using the `oci` provider. The `oci` provider uses the Kubernetes Cluster API Provider for OCI to perform the deployment. This is an implementation of the Kubernetes Cluster API. The Kubernetes Cluster API is implemented as Kubernetes Custom Resources (CRs), that are serviced by applications running in a Kubernetes cluster. The Kubernetes Cluster API has a large interface and is explained in the upstream documentation. For information on the Kubernetes Cluster API, see the Kubernetes Cluster API documentation. For information on the Cluster API implementation for OCI, see the Kubernetes Cluster API Provider for OCI documentation.

You can make custom installations of the Oracle Container Host for Kubernetes (OCK) image on arbitrary platforms. This means you can create a Kubernetes cluster using bare metal or other virtual instances, not provided explicitly by Oracle CNE. These installations are known as *Bring Your Own* (BYO) installations. You use the `byo` provider to perform these installations.

Configuration information used to create a cluster can be specified in several locations:

- Global defaults in the default configuration file, set in the `$HOME/.ocne/defaults.yaml` file.

- Kubernetes cluster configuration files. These files set the options for individual clusters and can be any name.
- Options provided with the `ocne` command.

For information on the default configuration file and the inheritance of each setting, see Oracle Cloud Native Environment: CLI, and for information on cluster configuration files, see Cluster Configuration Files.

To provide High Availability of control plane nodes, you can specify the location of an external load balancer. Or you can use an internal deployment of Keepalived and NGINX as the load balancer. When using the internal load balancer, specify the IP address that the Kubernetes API Server should use as the virtual IP address, or an IP address can be set automatically using the subnet of the control plane nodes.

In addition to the CLI, Oracle CNE also includes a web-based UI which can be used to manage the maintenance and installation of Kubernetes cluster resources, and applications. You can opt to install this, or install a headless cluster and only use the CLI.

Kubernetes applications are delivered and installed using an application catalog. A default Oracle catalog is installed when you create a cluster. You can install applications using both the CLI and the UI. You can also add other application catalogs, such as the Artifact Hub. For information on application catalogs, and applications, see Oracle Cloud Native Environment: Applications.

2

Cluster Configuration Files

Describes Kubernetes cluster configuration files in Oracle CNE.

Most deployments contain at least some details that are unique to that deployment. A cluster configuration file can be used to provide those customizations without specifying them on the command line.

Cluster configuration files set the options to customize a cluster. The contents of a cluster configuration file override any defaults set in the CLI default configuration file (`$HOME/.ocne/defaults.yaml`). For information on the default configuration file, see Oracle Cloud Native Environment: CLI.

A cluster configuration file is a YAML file, and can be anywhere on the localhost, for example, `$HOME/.ocne/mycluster.yaml`. Use this file by including the `--config` option when creating a cluster with the `ocne cluster start` command.

A cluster configuration file can be overridden by providing options on the command line when you use the `ocne cluster start` command to create a cluster.

Cluster Configuration File Options

Lists the options that can be included in cluster configuration files to customize Kubernetes clusters.

Many of the options in a cluster configuration file can also be set in the defaults file (`$HOME/.ocne/defaults.yaml`). If a setting is included in a cluster configuration file, it overrides the defaults file. A cluster configuration file can contain any of the following options:

`applications`
A list of applications to deploy to the target cluster after it's instantiated. The fields you can use are:

- `application`: The name of the application to install from a catalog.
- `name`: Sets a deployment name for the application. This is useful for applications that might benefit from having several installations in a single cluster.
- `catalog`: The name of the catalog from which to install the application. The default is `Oracle Cloud Native Environment Application Catalog`.
- `config`: Sets the URL of a configuration file for the application.

For example:

```
applications:
  - name: istio-ingress
  - name: istio-egress
  - name: prometheus
  - name: grafana
  - name: multus
```

or:

```
applications:
  - name: istio-ingress
    release: my-istio-ingress
    catalog: embedded
  - name: istio-egress
    release: my-istio-egress
    catalog: embedded
  - name: prometheus
    release: my-prometheus
    catalog: embedded
  - name: grafana
    release: my-grafana
    catalog: embedded
  - name: multus
    release: my-multus
    catalog: embedded
    config: git://gitlab.example.com/configs/multus.yaml
```

`autoStartUI`
Sets whether a tunnel to the Oracle CNE UI service is created when a cluster is instantiated, and starts the default browser to load the UI. For example:

```
autoStartUI: true
```

`bootVolumeContainerImage`
The container image registry and tag that contains an Oracle Container Host for Kubernetes (OCK) bootable image. The default is container-registry.oracle.com/olcne/ock:1.33. For example:

```
bootVolumeContainerImage: container-registry.oracle.com/olcne/ock:1.33
```

`catalogs`
A list of catalogs to deploy to the target cluster after it's instantiated. The `name` field indicates the name of the catalog. This is used by other Oracle CNE interfaces to refer to the catalog. The `uri` field sets the source of the catalog. Depending on the source, a pod might be deployed into the cluster that serves the catalog content. For example:

```
catalogs:
  - name: mycatalog
    uri: https://mycatalog.example.com/catalog
```

`clusterDefinition`
The path to a cluster configuration file. Provide this extra layer of configuration for clusters that use complex configuration that isn't provided by this default configuration file. For example:

```
clusterDefinition: mycluster.yaml
```

clusterDefinitionInline
Specifies in-line configuration options. Provide this extra layer of configuration for clusters that use complex configuration that isn't provided by this default configuration file. This option can't be used with `clusterDefinition`. For example:

```
clusterDefinitionInline: |
  key1: value1
  key2: value2
```

cni
The Container Networking Interface (CNI) provider to install when the cluster is instantiated. The value can be any CNI available with Oracle CNE, or `none` if another CNI is to be deployed either manually or using an application catalog.

> **Note**
>
> Multus can't be used as the primary CNI. Multus is available as an application in the default application catalog. If you install the Multus application, set this option to `none`.

For example:

```
cni: flannel
```

```
cni: none
```

communityCatalog
Sets whether the Artifact Hub application catalog is installed. If this is set to `true`, the catalog is installed. If this is set to `false`, the catalog isn't installed. The default is `false`. For example:

```
communityCatalog: true
```

controlPlaneNodes
Sets the number of control plane nodes to spawn. The default is `1`. For example:

```
controlPlaneNodes: 3
```

ephemeralCluster
Allows customization of any short-lived clusters that might be spawned to perform tasks that can't be completed on the host system. This is often used for changing boot OCK images or deploying Kubernetes Cluster API resources. The options you can use are:

name
The name of the cluster. For example:

```
ephemeralCluster:
  name: mycluster
```

preserve
Sets whether the ephemeral cluster is automatically deleted after the work is complete. The default is `false`, so ephemeral clusters are deleted after they're used. For example:

```
ephemeralCluster:
  preserve: true
```

node
Sets the configuration for the VMs. For example:

```
ephemeralCluster:
  node:
    cpus: 2
    memory: 4GB
    storage: 15GB
```

extraIgnition
The path to an Ignition file that includes extra Ignition information to include when creating a cluster, or joining nodes to a cluster. The Ignition information must comply with the Ignition specification v3.4.0, as listed in the upstream Ignition documentation, and written in YAML using the Butane Fedora CoreOS Specification v1.5.0, as described in the upstream Butane documentation. For example:

```
extraIgnition: /home/username/.ocne/ignition.ign
```

extraIgnitionInline
Extra Ignition information to include when creating a cluster, or joining nodes to a cluster. The Ignition information must comply with the Ignition specification v3.4.0, as listed in the upstream Ignition documentation, and written in YAML using the Butane Fedora CoreOS Specification v1.5.0, as described in the upstream Butane documentation. The format must be:

```
extraIgnitionInline: |
  key1: value1
  key2: value2
  ...
```

headless
Sets whether the Oracle CNE UI is installed. If this is set to `true`, the UI isn't installed. The default is `false`. For example:

```
headless: true
```

kubeApiServerBindPort
Sets the port on which the Kubernetes API Server is exposed. The default is `6443`. For example:

```
kubeApiServerBindPort: 6443
```

kubeApiServerBindPortAlt
Sets the port on which the Kubernetes API Server listens when deploying a Highly Available cluster using the Keepalived and NGINX load balancer. The default is `6444`. For example:

```
kubeApiServerBindPortAlt: 6444
```

kubeconfig
The path to the `kubeconfig` file to use for operations that require a running cluster. For example:

```
kubeconfig: /home/username/.kube/kubeconfig.utilitycluster
```

kubeProxyMode
The mode for `kube-proxy`. This can be set to either `iptables` or `ipvs`. The default is `iptables`. For example:

```
kubeProxyMode: ipvs
```

For more information on the `kube-proxy` modes, see the upstream Kubernetes documentation.

kubernetesVersion
This defines the Kubernetes version. The default is the latest version. For example:

```
kubernetesVersion: 1.33
```

loadBalancer
The hostname or IP address of an external load balancer for the Kubernetes API Server. Don't use this if the `virtualIp` option is used. For example:

```
loadBalancer: 100.100.1.1
```

name
The name of the cluster. The default is `ocne`. For example:

```
name: mycluster
```

osRegistry
Combined with `osTag`, this identifies an OSTree image in a container registry. It specifies the OSTree transport and the container registry URI.
Possible prefixes for the transport are:

```
ostree-image-signed
ostree-remote-image
ostree-unverified-image
ostree-unverified-registry
```

The default value is:

```
osRegistry: ostree-unverified-registry:container-registry.oracle.com/olcne/
ock-ostree
```

osTag
Combined with `osRegistry`, this identifies an OSTree image in a container registry. It specifies the tag for the image. For example:

```
osTag: 1.33
```

password
A hashed password for the OCK image user (`ocne`) to authenticate with cluster nodes. For example:

```
password: $6$jfkldjfsd$n1YMnpdxlGXO...
```

Surrounding the password with quotes is optional.
You can use the `openssl` utility to create a hashed password. For example, to generate a hashed password with the SHA512 algorithm and an automatic salt:

```
openssl passwd -6 -salt password
```

To generate a SHA512 hashed password using the provided salt phrase:

```
openssl passwd -6 -salt saltphrase password
```

podSubnet
The subnet to use for the pod network. The CNI is automatically configured to use this subnet. For example:

```
podSubnet: 10.244.0.0/16
```

provider
Sets the provider to use when creating a cluster. The options are:

- `byo`
- `libvirt` (the default)
- `none`
- `oci`
- `olvm`

For example:

```
provider: byo
```

providers
Specifies provider configuration options. For example:

```
providers:
  byo:
    options
  libvirt:
    options
  none:
    options
  oci:
    options
  olvm:
    options
```

The options for each provider are listed in:

- Bring Your Own Provider Options

  libvirt Provider Options
- OCI Provider Options
- Oracle Linux Virtualization Manager Provider Options

proxy
The proxy server information. This information is configured on the Kubernetes nodes. For example:

```
proxy:
  httpsProxy: http://myproxy.example.com:2138
  httpProxy: http://myproxy.example.com:2138

noProxy: .example.com,127.0.0.1,localhost,169.254.169.254,10.96.0.0/12,10.244.
0.0/16
```

quiet
Sets whether to reduce the messages printed by the `ocne` command. If this is set to `true`, the messages are reduced. If set to `false`, the messages aren't reduced. The default is `false`. For example:

```
quiet: true
```

registry
Sets the registry from which to provision container images. The default is `container-registry.oracle.com`. For example:

```
registry: myregistry.example.com
```

serviceSubnet
The subnet to use for the service network. The default is `10.96.0.0/12`. For example:

```
serviceSubnet: 10.96.0.0/12
```

sshPublicKey
The public key of an RSA key pair for the OCK image user (`ocne`). Paste the contents of the public key file.
For example:

```
sshPublicKey: |
    ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAA...
```

sshPublicKeyPath
The path to the public key of an RSA key pair for the OCK image user (`ocne`) to authenticate with cluster nodes.

```
sshPublicKeyPath: /home/username/.ssh/id_rsa.ocne
```

virtualIp
Sets the virtual IP address to use when using the built-in Keepalived and NGINX load balancer. If no IP address is provided, one is generated from the subnet of the control plane nodes. Don't use this if the `loadBalancer` option is used. For example:

```
virtualIp: 192.168.0.100
```

workerNodes
Sets the number of worker nodes to spawn. The default is `0`. For example:

```
workerNodes: 5
```

Bring Your Own Provider Options

The options for the `byo` provider are:

automaticTokenCreation
If set to `true`, any time a join token is required it's created automatically as part of the command. If it's set to `false`, the token must be created manually. For example:

```
providers:
  byo:
    automaticTokenCreation: false
```

networkInterface
Sets the network interface to which the CNI and other Kubernetes services bind. This option is required. For example:

```
providers:
  byo:
    networkInterface: enp1s0
```

libvirt Provider Options

The options for the `libvirt` provider are:

`controlPlaneNode`
Sets the configuration options for control plane nodes. To specify sizes, use:

- `M`: megabytes
- `G`: gigabytes
- `Mi`: mebibytes
- `Gi`: gibibytes

For example:

```
providers:
  libvirt:
    controlPlaneNode:
      cpu: 2
      memory: 16Gi
      storage: 8Gi
```

`network`
The name of the virtual network to use for domains. For example:

```
providers:
  libvirt:
    network: bridge-1
```

`sshKey`
The path to an SSH key to use for SSH connections. For example:

```
providers:
  libvirt:
    sshKey: /home/username/.ssh/id_rsa.ocne
```

`storagePool`
The name of the storage pool to use for OCK images. For example:

```
providers:
  libvirt:
    storagePool: mypool
```

`uri`
The default value for the libvirt connection URI. For example, for a local connection:

```
providers:
  libvirt:
    uri: qemu:///system
```

And for a remote connection:

```
providers:
  libvirt:
    uri: qemu+ssh://user@host/system
```

`workerNode`
Sets the configuration options for worker nodes. To specify sizes, use:

- `M`: megabytes
- `G`: gigabytes
- `Mi`: mebibytes
- `Gi`: gibibytes

For example:

```
providers:
  libvirt:
    workerNode:
      cpu: 2
      memory: 16Gi
      storage: 8Gi
```

OCI Provider Options

The options for the `oci` provider are:

`compartment`
The OCI compartment in which to deploy resources. This can be either the path to a compartment (for example, `mytenancy/mycompartment`), or the OCID of a compartment. For example:

```
providers:
  oci:
    compartment: OCID
```

`controlPlaneShape`
The name of the shape to use for compute instances when creating control plane nodes. For example:

```
providers:
  oci:
    controlPlaneShape:
      shape: VM.Standard.E4.Flex
      ocpus: 2
```

`imageBucket`
The OCID or name of a bucket to use to store OCK boot images when they're uploaded to OCI object storage. The default name is `ocne-images`. For example:

```
providers:
  oci:
    imageBucket: ocne-images
```

`images`
The OCI OCIDs of the OCK images to use as the initial disk image for any compute resources. Sets the options for the `amd64` and `arm64` architectures. For example:

```
providers:
  oci:
    images:
      amd64: OCID
      arm64: OCID
```

`kubeconfig`
The path to the `kubeconfig` file to use for the target management cluster. For example:

```
providers:
  oci:
    kubeconfig: /home/username/.kube/kubeconfig.mgmtcluster
```

`loadBalancer`
The OCIDs for subnets to use when provisioning OCI load balancers for default deployments. For example:

```
providers:
  oci:
    loadBalancer:
      subnet1: OCID
      subnet2: OCID
```

`namespace`
The Kubernetes namespace where the Kubernetes Cluster API resources are to be deployed.

```
providers:
  oci:
    namespace: mynamespace
```

`selfManaged`
Sets whether a cluster is self-managing. If set to `true`, the cluster contains the necessary controllers and resources to manage its own life cycle. If set to `false`, or not set, those resources remain in the initial administration cluster. For example:

```
providers:
  oci:
    selfManaged: true
```

`profile`
Sets the OCI CLI profile to use. This is the name of the profile in the OCI CLI configuration file. The default profile is `DEFAULT`. For example:

```
providers:
  oci:
    profile: MYTENANCY
```

vcn
The OCID of the Virtual Cloud Network to use when creating load balancers for default deployments.

```
providers:
  oci:
    vcn: OCID
```

workerShape
The name of the shape to use for compute instances when creating worker nodes. For example:

```
providers:
  oci:
    workerShape:
      shape: VM.Standard.E4.Flex
      ocpus: 2
```

Oracle Linux Virtualization Manager Provider Options

The options for the `olvm` provider are:

controlPlaneMachine
Sets the options for the control plane VM nodes. The options are:
`olvmNetwork`: The network settings for VMs, which includes:

- `networkName`: The Oracle Linux Virtualization Manager network name. This must exist in the Oracle Linux Virtualization Manager instance.
- `vnicName`: The Oracle Linux Virtualization Manager VNIC name. This is optional.
- `vnicProfileName`: The Oracle Linux Virtualization Manager VNIC profile name. This must exist in the Oracle Linux Virtualization Manager instance.

`olvmOvirtClusterName`: The name of the Oracle Linux Virtualization Manager cluster. This must exist in the data center.
`virtualMachine`: Sets the options for VMs. The options are:

- `cpu`: Information about the CPU. The options are:
  - `topology`:
    - `architecture`: The architecture of the CPU. The default is `x86_64`.
    - `cores`: The number of cores. The default is `2`.
    - `sockets`: The number of sockets. The default is `2`.
    - `threads`: The number of threads. The default is `1`.
- `memory`: The size of memory for VMs. The default is `7GB`.
- `network`: Sets the network options for VMs. The options are:
  - `gateway`: The IP address of the network gateway for VMs.
  - `interface`: The network interface name for VMs. This is optional. The default is `enp1s0`.

- `interfaceType`: The network interface type for VMs. This is optional. The default is `virtio`.
- `ipv4`: The IPv4 network configuration for VMs. This is mandatory. The options are:
  - `subnet`: The IPv4 subnet.
  - `ipAddresses`: The IPv4 addresses that can be used. This is a comma separated list, with any combination of CIDRs, IP ranges (inclusive), and IPs. Spaces after commas are optional. For example:

    ```
    192.0.2.0/24, 192.0.2.10-192.0.2.20, 192.0.2.21
    ```

- `ipv6`: The IPv6 network configuration for VMs. This is optional. The options are:
  - `ipAddresses`: The IPv6 addresses that can be used. This is a comma separated list. Spaces after commas are optional. This option is required if IPv6 is used. For example:

    ```
    2001:db8:0f01::1-2001:db8:0f01::9/64
    ```

`vmTemplateName`: The name of the Oracle Linux Virtualization Manager VM template. This must exist in the Oracle Linux Virtualization Manager instance.
For example:

```
providers:
  olvm:
    controlPlaneMachine:
      olvmOvirtClusterName: Default
      vmTemplateName: ock-1.33
      olvmNetwork:
        networkName: kvm-vlan
        vnicName: nic-1
        vnicProfileName: kvm-vlan
      virtualMachine:
        memory: "8GB"
        cpu:
          topology:
            cores: 8
            sockets: 10
            threads: 2
        network:
          gateway: 1.2.3.1
          interface: enp1s0
          interfaceType: virtio
          ipv4:
            subnet: 192.0.0.0/24
            ipAddresses: 192.0.2.0/24, 192.0.2.10-192.0.2.20, 192.0.2.21
          ipv6:
            ipAddresses: 2001:db8:0f01::1-2001:db8:0f01::9/64
```

namespace
The Kubernetes namespace where the Kubernetes Cluster API resources are to be deployed in the management cluster. The default is `olvm`. For example:

```
providers:
  olvm:
    namespace: mynamespace
```

olvmDatacenterName
The name of the Oracle Linux Virtualization Manager data center instance. This option is mandatory. For example:

```
providers:
  olvm:
    olvmDatacenterName: myolvm
```

olvmOCK
Sets the information needed to upload the Oracle Linux Virtualization ManagerOCK image using the `ocne image upload` command, and includes:

- `diskName`: The name of the disk to be created in the storage domain when the image is upload. This is the disk name you specify when you create a VM template.
- `diskSize`: The provisioned virtual disk size name to be used for the disk created in the storage domain. This is the disk space to be allocated for the VM, regardless of the size of the image, on disk. For example, the image might be 2.5GB, but the provisioned size could be 16GB.
- `storageDomainName`: The name of an existing Oracle Linux Virtualization Manager storage domain where the image is to be uploaded.

For example:

```
providers:
  olvm:
    ovirtOCK:
      storageDomainName: olvm-data
      diskName: ock-1.33
      diskSize: 16GB
```

olvmOvirtAPIServer
Sets the options for the Oracle Linux Virtualization Manager cluster inside the data centre. This information is used to connect to the Oracle Linux Virtualization Manager oVirt REST API server. The options are:
`caConfigMap`: Sets the name and namespace of the ConfigMap containing the CA Certificate. This is optional, and includes:

- `name`: The name of the ConfigMap. The default is `olvm-ca`.
- `namespace`: The namespace for the ConfigMap. The default is `olvm`.

`credentialsSecret`: Sets the name and namespace of the Secret containing the credentials needed to communicate with the Oracle Linux Virtualization Manager server, which includes:

- `name`: The name of the Secret. The default is `olvm-creds`.
- `namespace`: The namespace for the Secret. The default is `olvm`.

`insecureSkipTLSVerify`: A Boolean to set whether to skip the validity check for the server's certificate when connecting to the Oracle Linux Virtualization Manager server Set this to `true` if the CA isn't needed or used. This is optional. The default is `false`.
`serverCA`: The Oracle Linux Virtualization Manager root Certificate Authority (CA), inline. This is mutually exclusive with the `serverCAPath` option. For information on obtaining the CA Certificate, see the oVirt upstream documentation.
`serverCAPath`: The path to the Oracle Linux Virtualization Manager root Certificate Authority (CA) Certificate. This is mutually exclusive with the `serverCA` option.
`serverURL`: The URL to connect to the Oracle Linux Virtualization Manager data centre. This option is mandatory.
For example:

```
providers:
  olvm:
    olvmOvirtAPIServer:
      serverURL: https://my.example.com/ovirt-engine
      serverCAPath: "/home/username/olvm/ca.crt"
      credentialsSecret:
        name: olvm-creds
        namespace: olvm
      caConfigMap:
        name: olvm-ca
        namespace: olvm
      insecureSkipTLSVerify: false
```

`ovirtCsiDriver`
Sets the configuration for the `ovirt-csi-driver`. These settings are optional. By default, the `ovirt-csi-driver` driver is automatically installed, along with the required namespace, credential Secret, CA Certificate, ConfigMap, and CsiDriver resources. The configuration options are:
`caConfigmapName`: Sets the name of the ConfigMap containing the CA Certificate.
`controllerPluginName`: Sets the name of the deployment for the controller plugin, which is part of the driver. The default is `ovirt-csi-controller`.
`credsSecretName`: Sets the name of the Secret containing the credentials needed to communicate with the Oracle Linux Virtualization Manager server. The default is `ovirt-csi-creds`.
`csiDriverName`: Sets the name of the CsiDriver. This name is used when you create a StorageClass (the value of the `provisioner` field). The default is `csi.ovirt.org`.
`install`: A Boolean to set whether to install the driver and the required resources. The default is `true`.
`namespace`: Sets the namespace where the driver, and all related resources, are created. The default is `ovirt-csi`.
`nodePluginName`: Sets the name of the daemonset for the node plugin, which is part of the driver. The default is `ovirt-csi-node`.
For example:

```
providers:
  olvm:
    ovirtCsiDriver:
      install: true
      caConfigmapName: ovirt-csi-ca.crt
      controllerPluginName: ovirt-csi-controller
      credsSecretName: ovirt-csi-creds
```

```
    csiDriverName: csi.ovirt.org
    namespace: ovirt-csi
    nodePluginName: ovirt-csi-node
```

workerMachine
Sets the options for the worker VM nodes. The options are:
olvmNetwork: The network settings for VMs, which includes:

- networkName: The Oracle Linux Virtualization Manager network name. This must exist in the Oracle Linux Virtualization Manager instance.
- vnicName: The Oracle Linux Virtualization Manager VNIC name. This is optional.
- vnicProfileName: The Oracle Linux Virtualization Manager VNIC profile name. This must exist in the Oracle Linux Virtualization Manager instance.

olvmOvirtClusterName: The name of the Oracle Linux Virtualization Manager cluster. This must exist in the data center.
virtualMachine: Sets the options for VMs. The options are:

- cpu: Information about the CPU. The options are:
  - topology:
    - architecture: The architecture of the CPU. The default is x86_64.
    - cores: The number of cores. The default is 2.
    - sockets: The number of sockets. The default is 2.
    - threads: The number of threads. The default is 1.
- memory: The size of memory for VMs. The default is 16GB.
- network: Sets the network options for VMs. The options are:
  - gateway: The IP address of the network gateway for VMs.
  - interface: The network interface name for VMs. This is optional. The default is enp1s0.
  - interfaceType: The network interface type for VMs. This is optional. The default is virtio.
  - ipv4: The IPv4 network configuration for VMs. This is mandatory. The options are:
    - subnet: The IPv4 subnet.
    - ipAddresses: The IPv4 addresses that can be used. This is a comma separated list, with any combination of CIDRs, IP ranges (inclusive), and IPs. Spaces after commas are optional. For example:

      ```
      192.0.2.0/24, 192.0.2.10-192.0.2.20, 192.0.2.21
      ```

  - ipv6: The IPv6 network configuration for VMs. This is optional. The options are:

* `ipAddresses`: The IPv6 addresses that can be used. This is a comma separated list. Spaces after commas are optional. This option is required if IPv6 is used. For example:

  ```
  2001:db8:0f01::1-2001:db8:0f01::9/64
  ```

`vmTemplateName`: The name of the Oracle Linux Virtualization Manager VM template. This must exist in the Oracle Linux Virtualization Manager instance.
For example:

```
providers:
  olvm:
    workerMachine:
      olvmOvirtClusterName: Default
      vmTemplateName: ock-1.33
      olvmNetwork:
        networkName: kvm-vlan
        vnicName: nic-1
        vnicProfileName: kvm-vlan
      virtualMachine:
        memory: "16GB"
        cpu:
          topology:
            cores: 6
            sockets: 6
            threads: 3
        network:
          gateway: 1.2.3.1
          interface: enp1s0
          interfaceType: virtio
          ipv4:
            subnet: 192.0.0.0/24
            ipAddresses: 192.0.2.0/24, 192.0.2.21-192.0.2.30, 192.0.2.32
          ipv6:
            ipAddresses: 2001:db8:0f01::1-2001:db8:0f01::9/64
```

Configuration File Examples

Provides example configuration files.

Example 2-1 Set some default options to create remote libvirt clusters

This example uses a specific SSH key to connect to a remote system to create clusters, includes the proxy server configuration, and doesn't install the UI.

```
providers:
  libvirt:
    uri: qemu+ssh://myuser@host.example.com/system
    sshKey: /home/username/.ssh/id_rsa.ocne
proxy:
  httpsProxy: http://myproxy.example.com:2138
  httpProxy: http://myproxy.example.com:2138

noProxy: .example.com,127.0.0.1,localhost,169.254.169.254,10.96.0.0/12,10.244.
```

```
0.0/16
headless: true
```

Example 2-2 Set some default options to create local libvirt clusters

This example sets several options, including the sizes for the Kubernetes nodes in a libvirt cluster.

```
provider: libvirt
name: mycluster
workerNodes: 2
controlPlaneNodes: 1
providers:
  libvirt:
    controlPlaneNode:
      cpu: 2
      memory: 8Gi
      storage: 20Gi
    workerNode:
      cpu: 2
      memory: 8Gi
      storage: 20Gi
```

3

Oracle Container Host for Kubernetes Image

Describes the Oracle Container Host for Kubernetes (OCK) image used to create nodes in a Kubernetes cluster.

Oracle CNE includes a CLI that can manage the life cycle of Kubernetes clusters, using OSTree based container images. The container image includes both the host Oracle Linux OS, and the Kubernetes software distribution. The image is deployed to hosts or Virtual Machines (VMs) to create nodes in a Kubernetes cluster. This image is referred to in this documentation as the Oracle Container Host for Kubernetes (OCK) image.

The OCK image is distributed on the Oracle Container Registry in the following formats:

Bootable image
This is a container image in the Qcow2 format, available at:
container-registry.oracle.com/olcne/ock
The bootable image contains a single VM image in the Qcow2 format, and is used to create boot media for virtualized platforms. This image is used as the boot media for clusters created with the libvirt and Kubernetes Cluster API providers (the OCI and Oracle Linux Virtualization Manager providers).
By default, the image is configured to work with the libvirt provider. A conversion of the boot image to the appropriate format for OCI and Oracle Linux Virtualization Manager is done when you upload the image.

OSTree image
This is an OSTree commit based container image, available at:
container-registry.oracle.com/olcne/ock-ostree
This image is used as the basis for an OSTree archive for customized installations using the Bring Your Own provider.
This image is also used for updating cluster nodes to stage patch updates, and to update to the next Kubernetes minor release.
For information on OSTree containers, see the upstream OSTree documentation.

Both images use the container label for the Kubernetes version they match, for example, 1.33.

OCK Image User

Describes the `ocne` user set up in the OCK image, and how to configure it.

To interact directly with a node, use its console. For information on connecting to a node's console, see Accessing a Node's OS Console.

If a node fails and its console is inaccessible, you can SSH into the node as the `ocne` user. The `ocne` user is predefined in the OCK image.

> **⚠ Caution**
>
> When you SSH into a node, it returns the node's `kubeconfig` file and the SSH connection immediately closes. Full access to the node is only possible using its console.

By default, the `ocne` user authenticates with the cluster nodes through SSH using an RSA public key in `$HOME/.ssh/id_rsa.pub`.

To create an SSH2 RSA key pair, run the following command:

```
ssh-keygen
```

Follow the prompts to generate and store the RSA key pair:

```
Generating public/private rsa key pair.
      Enter file in which to save the key (/home/user/.ssh/id_rsa): <Enter>
      Created directory '/home/user/.ssh'.
      Enter passphrase (empty for no passphrase): password
      Enter same passphrase again: password
      Your identification has been saved in /home/user/.ssh/id_rsa.
      Your public key has been saved in /home/user/.ssh/id_rsa.pub.
      The key fingerprint is:
      5e:d2:66:f4:2c:c5:cc:07:92:97:c9:30:0b:11:90:59 user@host01
      The key's randomart image is:
      +--[ RSA 2048]----+
      |      .=Eo++.o   |
      |      o  ..B=.   |
      |          o.= .  |
      |         o + .   |
      |        S * o    |
      |       . = .     |
      |        .        |
      |       .         |
      |                 |
      +-----------------+
```

To authenticate using a different method, configure *one* of the following options in either the CLI defaults configuration file, or a cluster configuration file:

`password`
A hashed password for the OCK image user (`ocne`) to authenticate with cluster nodes. For example:

```
password: $6$jfkldjfsd$n1YMnpdxlGXO...
```

Surrounding the password with quotes is optional.

You can use the `openssl` utility to create a hashed password. For example, to generate a hashed password with the SHA512 algorithm and an automatic salt:

```
openssl passwd -6 -salt password
```

To generate a SHA512 hashed password using the provided salt phrase:

```
openssl passwd -6 -salt saltphrase password
```

sshPublicKey
The public key of an RSA key pair for the OCK image user (`ocne`). Paste the contents of the public key file.
For example:

```
sshPublicKey: |
    ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAA...
```

sshPublicKeyPath
The path to the public key of an RSA key pair for the OCK image user (`ocne`) to authenticate with cluster nodes.

```
sshPublicKeyPath: /home/username/.ssh/id_rsa.ocne
```

For information about these configuration file options, see Cluster Configuration Files and Oracle Cloud Native Environment: CLI.

Custom Images

Describes custom OCK images used to create Kubernetes cluster nodes.

To create custom images, use the `ocne image create` command. The resulting image is in the appropriate format for the target platform and saved to the `$HOME/.ocne/images` directory on the localhost. Use the `ocne image upload` command to upload the image to a location where you can use the image to create VMs.

Creating images requires access to a Kubernetes cluster. Any running cluster can be used. To specify which cluster to use, set the `KUBECONFIG` environment variable, or use the `--kubeconfig` option of `ocne` commands. If no cluster is available, an ephemeral cluster is created automatically using the `libvirt` provider, with the default configuration.

Image conversion requires a significant amount of space. We recommend you allocate at least 20 G of storage to any cluster nodes. You can set the storage for the ephemeral and libvirt clusters in a configuration file, for example, you could add the following to the CLI defaults file:

```
ephemeralCluster:
  node:
    storage: 20G
providers:
  libvirt:
    workerNode:
      storage: 20G
    controlPlaneNode:
      storage: 20G
```

OSTree Archive Images

Describes custom images in OSTree archive format to create Kubernetes nodes in Bring Your Own clusters.

Custom installations of OCK are done using the Anaconda and Kickstart automatic installation options of Oracle Linux. Anaconda requires OSTree content to be available in a particular format to install onto the root file system of the target host (an OSTree archive served over HTTP). The OCK container image used for updates isn't in this format. To use the content with Anaconda, it must be converted to an archive and made available using an HTTP server.

Images can be created using the `ocne image create` command with the `--type ostree` option. The resulting container image is stored in the Open Container Initiative archive format and can be imported to the local cache or pushed to a container registry. Building the container image pulls a base image from within the Kubernetes cluster (this is either the existing cluster set using a `kubeconfig` file, or an ephemeral cluster that's started for this purpose).

By default, the image is created for the architecture of the system where the command is run. Images can be created for other architectures using the `--arch` option.

Typically, OSTree archive images are copied to a container registry. You can copy the image to any target that works with the Open Container Initiative transports and formats. See `containers-transports(5)` for available options.

4

Proxy Servers

Describes the configuration options for deploying a Kubernetes cluster in an environment with a proxy server.

If a proxy server is included in the deployment environment, proxy server configuration might be needed for communication between:

- CLI to the Oracle Container Registry.
- CLI to the Kubernetes API Server endpoint.
- CLI to the Kubernetes Cluster API management cluster (management cluster) Kubernetes API Server endpoint.
- CLI to the cloud service gateway.
- Management cluster to Oracle Container Registry.
- Kubernetes Cluster API cloud controllers (cloud controllers) to the Internet.
- Cloud controllers to the Kubernetes API Server endpoint.
- Cloud controllers to the cloud service gateway.

No proxy configuration might need to be set for:

- Cloud controllers to the Kubernetes service network.
- Cloud controllers to the Kubernetes pod network.

Proxy server configuration can be set in:

- Environment variables on the host where the CLI is installed.
- The default configuration file (`$HOME/.ocne/defaults.yaml`).
- Cluster configuration files.
- Kubernetes applications.

Setting a Proxy Server for the CLI

Set up the proxy configuration for the Oracle CNE CLI.

If a proxy server is part of the environment in which a Kubernetes cluster is to be deployed, you might need to set up the proxy server configuration for the CLI. This configuration might be required for the CLI to access the Oracle Container Registry and the Kubernetes API Server. This must be performed on the host that has the CLI installed.

1. Set up the HTTPS_PROXY configuration.

   Configure the host so container images can be pulled from the Oracle Container Registry using HTTPS. Use the format:

   ```
   export HTTPS_PROXY=proxy_server:port
   ```

For example:

```
export HTTPS_PROXY=https://proxy.example.com:3128
```

2. Set up the NO_PROXY configuration.

   Configure the host so the CLI can communicate with the Kubernetes API Server. Use the format:

   ```
   export NO_PROXY=exclusion_list
   ```

   For example:

   ```
   export NO_PROXY=.example.com,127.0.0.1,localhost,169.254.169.254
   ```

   The `169.254.169.254` IP address is reserved by many cloud providers, including OCI, as the endpoint for compute instances to access their own metadata. For more information on this IP address and how it's used, see the OCI documentation.

Setting a Proxy Server for the Kubernetes Cluster

Set up the proxy configuration for the Kubernetes cluster.

If a proxy server is part of the environment in which a Kubernetes cluster is to be deployed, you might need to set up the proxy server configuration for the cluster. This configuration is set in the default configuration file (`$HOME/.ocne/defaults.yaml`), or in a cluster configuration file.

Use a configuration file (either the default configuration file, or a cluster configuration file) to set the proxy server information for the cluster. This information is used to set up CRI-O on the Kubernetes nodes.

Use the format:

```
proxy:
  httpsProxy: proxy_server:port
  httpProxy: proxy_server:port
  noProxy: exclusion_list
```

For example:

```
proxy:
  httpsProxy: http://myproxy.example.com:2138
  httpProxy: http://myproxy.example.com:2138

noProxy: .example.com,127.0.0.1,localhost,169.254.169.254,10.96.0.0/12,10.244.
0.0/16
```

The `169.254.169.254` IP address is reserved by many cloud providers, including OCI, as the endpoint for compute instances to access their own metadata. For more information on this IP address and how it's used, see the OCI documentation.

The `10.96.0.0/12` CIDR is the default range for the Kubernetes service subnet. The `10.244.0.0/16` CIDR is the default range for the Kubernetes pod network.

Setting a Proxy Server for the UI

Set the proxy server configuration to use with the UI.

If you have installed a non default application catalog, such as the Artifact Hub catalog, and the deployment network uses a proxy server, you might need to update the `ui` application to include the proxy server information. Use the same proxy information used in a configuration file to create the cluster.

1. Create a proxy configuration file.

   Create a YAML file that includes the proxy information. Use the format:

   ```
   env:
     - name: https_proxy
       value: proxy_server:port
     - name: http_proxy
       value: proxy_server:port
     - name: no_proxy
       value: exclusion_list
   ```

   For example:

   ```
   env:
     - name: https_proxy
       value: http://myproxy.example.com:2138
     - name: http_proxy
       value: http://myproxy.example.com:2138
     - name: no_proxy
       value:
   ".example.com,127.0.0.1,localhost,169.254.169.254,10.96.0.0/12,10.244.0.0/1
   6"
   ```

2. Update the `ui` application.

   Use the `ocne application update` command to update the `ui` application in the `ocne-system` namespace. Use the syntax:

   ```
   ocne application update
   {-b|--built-in-catalog}
   [{-c|--catalog} name]
   [{-n|--namespace} namespace]
   {-r|--release} name
   [--reset-values]
   [{-u|--values} URI]
   [{-v|--version} version]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne application update --namespace ocne-system --release ui --values
   myproxy.yaml
   ```

5
libvirt Provider

Use the `libvirt` provider to create KVM based Kubernetes clusters with libvirt.

The `libvirt` provider is the default cluster provider, and can be used to provision Kubernetes clusters using Kernel-based Virtual Machines (KVM). The default KVM stack includes libvirt, and is included, by default, with Oracle Linux.

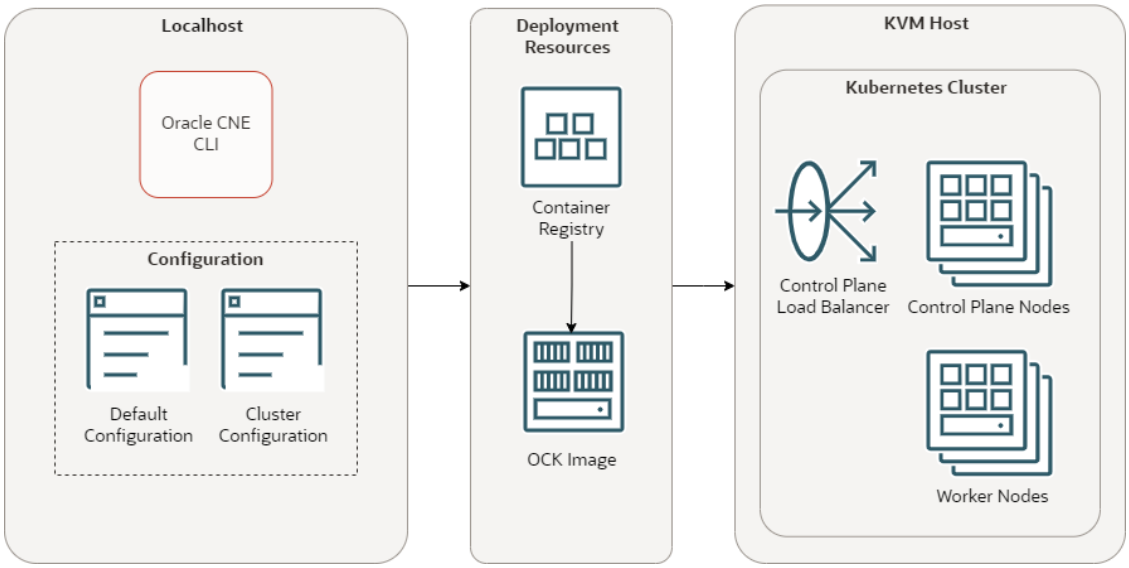
> **Note**
>
> We recommend the Oracle KVM stack as this KVM version offers many more features for Oracle Linux systems. For information on the Oracle KVM stack and libvirt, see the Oracle Linux 9: KVM User's Guide or the Oracle Linux 8: KVM User's Guide.

The system used to create libvirt clusters must be a 64-bit x86 or 64-bit ARM system running Oracle Linux 8 or 9, and include the Unbreakable Enterprise Kernel Release 7 (UEK R7).

The `libvirt` provider provisions Kubernetes clusters using libvirt on a single host, and is useful for creating and destroying Kubernetes clusters for testing and development. While the `libvirt` provider can be used for test and development clusters, it does deploy a production worthy cluster configuration.

> **Caution**
>
> As all libvirt cluster nodes are running on a single host, be aware that if the host running the cluster goes down, so do all the cluster nodes.

Figure 5-1 libvirt Cluster Architecture

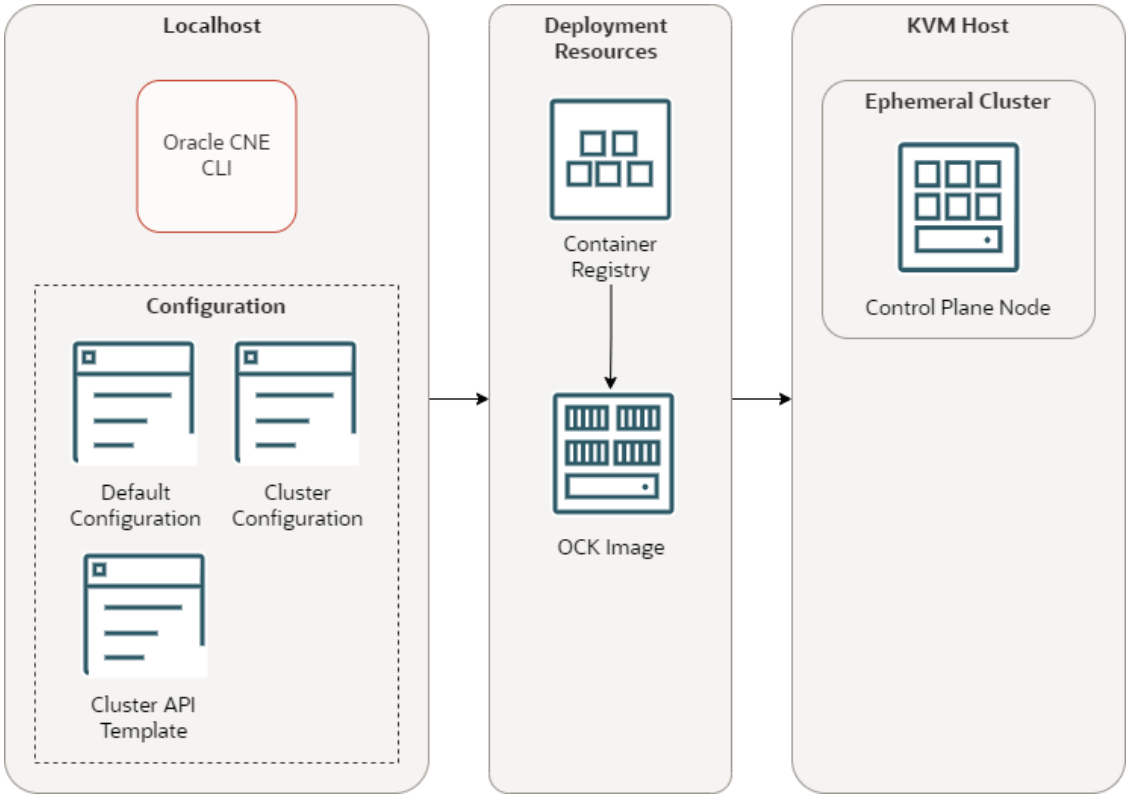


The libvirt cluster architecture has the following components:

- **CLI**: The CLI used to create and manage Kubernetes clusters. The `ocne` command.
- **Default configuration**: A YAML file that contains configuration for all `ocne` commands.
- **Cluster configuration**: A YAML file that contains configuration for a specific Kubernetes cluster.
- **Container registry**: A container registry used to pull the images used to create nodes in a Kubernetes cluster. The default is the Oracle Container Registry.
- **OCK image**: The OCK image pulled from the container registry, which is used to create Kubernetes nodes.
- **Control plane load balancer**: A load balancer used for High Availability (HA) of the control plane nodes.
- **Control plane nodes**: Control plane nodes in a Kubernetes cluster.
- **Worker nodes**: Worker nodes in a Kubernetes cluster.

The `libvirt` provider is also used to provision Kubernetes clusters when using some CLI commands. This cluster type is often referred to as an *ephemeral cluster*. An ephemeral cluster is a single node cluster that lives for a short time and is created and destroyed as needed by the CLI. An existing cluster can also be used as an ephemeral cluster by including the location of a `kubeconfig` file as an option with CLI commands.

Figure 5-2 libvirt Ephemeral Cluster



The ephemeral cluster architecture has the following components:

- **CLI**: The CLI used to create and manage Kubernetes clusters. The `ocne` command.
- **Default configuration**: A YAML file that contains configuration for all `ocne` commands.
- **Cluster configuration**: A YAML file that contains configuration for a specific Kubernetes cluster.
- **Cluster API template**: A YAML file that contains Cluster Resources for the Kubernetes Cluster API to create a cluster.
- **Container registry**: A container registry used to pull the images used to create nodes in a Kubernetes cluster. The default is the Oracle Container Registry.
- **OCK image**: The OCK image pulled from the container registry, which is used to create Kubernetes nodes.
- **Ephemeral cluster**: A temporary Kubernetes cluster used to perform a CLI command. The default for this is a single node cluster created with the libvirt provider on the localhost. This might also be an external cluster.

Single and multi node clusters can be created on Oracle Linux 8 and 9, on both 64-bit x86 and 64-bit ARM systems. Because all cluster nodes run on a single host, it's not possible to create hybrid clusters. However, it's possible to use an ARM system to create a remote cluster on x86 hardware and, conversely, x86 hardware can be used to create a remote cluster on ARM.

The `libvirt` provider requires the target system to be running libvirt and requires that the user be configured to have access to libvirt. Oracle CNE implements a libvirt connection using the legacy single-socket client. If local libvirt clusters are created, the UNIX domain socket is used.

To create Kubernetes clusters on a remote system, enable a remote transport mechanism for libvirt. We recommend you set up SSH key-based authentication to the remote system as a normal user, and that you configure the user with the privilege to run libvirt. You can, however, use any of the libvirt remote transport options. For more information on libvirt remote transports, see the upstream libvirt documentation.

Most remote cluster deployments leverage the `qemu+ssh` transport, which uses SSH to tunnel the UNIX domain socket back to the CLI. Oracle CNE doesn't configure the libvirt transports or system services. This must be set up correctly, according to the documentation for the OS.

Clusters created with the `libvirt` provider create a tunnel so the cluster can be accessed through a port on the host where the cluster is deployed. The port range starts at 6443 and increments from there. As clusters are deleted, the ports are freed. If a cluster is created on a remote system, ensure a range of ports are accessible through the system firewall, starting at 6443.

> ⚠ **Caution**
>
> You can disable the firewall in a testing environment, however we don't recommend this for production systems.

Use the `ocne cluster start` command to create a Kubernetes cluster using the `libvirt` provider. As this provider is the default, you don't need to specify the provider type. For example:

```
ocne cluster start
```

This command creates a single node cluster using all the default options, and installs the UI and application catalog.

You can add extra command line options to the `ocne cluster start` command to set up the cluster with non default settings, such as the number of control plane and worker nodes. For information on these command options, see Oracle Cloud Native Environment: CLI.

You can also customize the default settings by adding options to the default configuration file, or a configuration file specific to the cluster you want to create. For information on these configuration files, see Cluster Configuration Files and Oracle Cloud Native Environment: CLI.

For clusters started on systems with access to privileged libvirt instances, two `kubeconfig` files are created when you create a cluster, one for access to the local cluster, and one that can be used on the remote cluster host.

Setting Up the libvirt Provider

Set up an Oracle Linux host to create Kubernetes clusters using the `libvirt` provider.

Clusters can be created on the localhost, or on a remote system. Perform these steps on the system to be used to create the cluster, whether that's the localhost for local clusters, or on a remote host if you're creating clusters on a remote system.

By default, KVM is built into the Oracle Linux kernel. You can use the default KVM stack, which includes libvirt. We recommend you use the Oracle KVM stack which is available in Oracle Linux 8 or 9 with the Unbreakable Enterprise Kernel (UEK). For Oracle Linux 9, the latest UEK Release 7 (UEK R7) must be installed. For Oracle Linux 8, UEK R6 or UEK R7 must be installed.

> ⚠ **Caution**
>
> Existing Virtual Machines created with one KVM stack might not be compatible, and might not start, after switching to another KVM stack.

For more information on installing and configuring KVM, see the Oracle Linux 9: KVM User's Guide or the Oracle Linux 8: KVM User's Guide.

1. (Optional) Install the Oracle KVM stack.
   - Oracle Linux 9:

     If you have an existing installation of the default KVM stack, remove it:

     ```
     sudo dnf remove -y libvirt qemu-kvm edk2
     ```

     Install the Oracle KVM stack:

     ```
     sudo dnf config-manager --enable ol9_kvm_utils
     sudo dnf group install -y "Virtualization Host"
     sudo dnf install -y virt-install virt-viewer
     ```

     Start the virtualization daemons.

     ```
     for drv in qemu network nodedev nwfilter secret storage interface
     proxy
       do
        sudo systemctl enable virt${drv}d.service
     ```

```
  sudo systemctl enable virt${drv}d{,-ro,-admin}.socket
  sudo systemctl start virt${drv}d{,-ro,-admin}.socket
 done
```

- Oracle Linux 8:

  If you have an existing installation of the default KVM stack, remove it:

  ```
  sudo dnf module remove -y virt --all
  sudo dnf module reset virt
  ```

  Install the Oracle KVM stack:

  ```
  sudo dnf config-manager --enable ol8_kvm_appstream
  sudo dnf module enable virt:kvm_utils3
  sudo dnf module install -y virt:kvm_utils3
  ```

  Enable and start the `libvirtd.service`:

  ```
  sudo systemctl enable --now libvirtd.service
  ```

2. Validate the host.

   Validate the host is set up for hardware virtualization, and can be used as a KVM host:

   ```
   virt-host-validate qemu
   ```

3. Configure the user.

   Configure the user to have privileged access to libvirt, add the user to the `libvirt` and `qemu` groups.

   ```
   sudo usermod -a -G libvirt,qemu $USER
   ```

   To enable the change to the user, log out, and log back into the host or terminal session.

4. (Optional) Open a range of ports in the firewall.

   If you're installing libvirt on a remote host, open a series of firewall ports so you can access nodes in the cluster from the localhost. You don't need to do this if you're installing libvirt on the localhost. Use the format:

   ```
   sudo firewall-cmd --add-port 6443-endrange/tcp
   sudo firewall-cmd --add-port 6443-endrange/tcp --permanent
   ```

   Replace *endrange* with the highest port number you want to open. For example, to open 20 ports, use:

   ```
   sudo firewall-cmd --add-port 6443-6463/tcp
   sudo firewall-cmd --add-port 6443-6463/tcp --permanent
   ```

   Restart `firewalld.service`

   ```
   sudo systemctl restart firewalld.service
   ```

Creating a libvirt Cluster

Create a Kubernetes cluster using the `libvirt` provider.

1. Set up the host to provision clusters using the `libvirt` provider.

   See Setting Up the libvirt Provider.

2. (Optional) Set up a cluster configuration file.

   A cluster configuration contains the cluster specific information to use when creating the cluster. Use a cluster configuration file to override cluster defaults, or you can use `ocne cluster start` command options to configure a cluster. A cluster configuration file might include:

   ```
   provider: libvirt
   name: mycluster
   workerNodes: 2
   controlPlaneNodes: 1
   providers:
     libvirt:
       controlPlaneNode:
         cpu: 2
         memory: 8Gi
         storage: 20Gi
       workerNode:
         cpu: 2
         memory: 8Gi
         storage: 20Gi
   ```

   For information about cluster configuration files, see Cluster Configuration Files.

3. Create a libvirt cluster.

   Use the `ocne cluster start` command to create a cluster. The syntax is:

   ```
   ocne cluster start
   [{-u|--auto-start-ui} {true|false}]
   [{-o|--boot-volume-container-image} URI]
   [{-C|--cluster-name} name]
   [{-c|--config} path]
   [{-n|--control-plane-nodes} integer]
   [{-i|--key} path]
   [--load-balancer address]
   [{-P|--provider} provider]
   [{-s|--session} URI]
   [{-v|--version} version]
   [--virtual-ip IP]
   [{-w|--worker-nodes} integer]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

When you create a cluster on a remote system, include the session information in the format `qemu+ssh://user@host/system` where *host* is the name or IP address of the remote system. For example:

```
--session qemu+ssh://myuser@myhost.example.com/system
```

Example 5-1 Create a default cluster using the `libvirt` provider

To create a libvirt cluster, using all default settings:

```
ocne cluster start
```

Example 5-2 Create a `libvirt` cluster using a configuration file

To create a `libvirt` cluster using a configuration file:

```
ocne cluster start --config myconfig.yaml
```

Example 5-3 Create a `libvirt` cluster with specified nodes and virtual IP

To create a cluster with a specified number of worker and control plane nodes, and a virtual IP address:

```
ocne cluster start --control-plane-nodes 3 --worker-nodes 5 --virtual-ip
192.168.0.100
```

Example 5-4 Create a remote `libvirt` cluster using a configuration file

To create a cluster on a remote host using a configuration file:

```
ocne cluster start --session qemu+ssh://myuser@myhost.example.com/system --
config myconfig.yaml
```

Connecting to a Cluster

Use the `kubectl` package to connect to a Kubernetes cluster created with the `libvirt` provider.

After creating a Kubernetes cluster, two Kubernetes configuration (`kubeconfig`) files are created to connect to the cluster using the `kubectl` command.

One file provides direct access to the true Kubernetes API server endpoint. This `kubeconfig` file is saved as `$HOME/.kube/kubeconfig.`*`cluster_name`*`.local`. You can use this file to access a cluster created on the localhost, or to access a cluster created on a remote libvirt host.

The second `kubeconfig` file provides access to a dedicated tunnel implemented with SLiRP to access the cluster on remote systems. This file is saved as `$HOME/.kube/kubeconfig.`*`cluster_name`*`.vm`. If the cluster is started on a remote system, and you want to log into that remote system to access the cluster, you need to copy this second `kubeconfig` to the remote system.

Install `kubectl` on the host on which you want to access the cluster, either the localhost, or the remote libvirt system.

These steps, unless explicitly mentioned, are to be performed on the host on which you want to access the cluster (either the localhost or a remote libvirt system).

1. (Optional) Copy the `kubeconfig` file.

   If you created the cluster on a remote system, and you want to log in to the remote system to access the cluster, copy the `kubeconfig` file ending in `.vm` from the localhost to the remote system. The file is available on the localhost at:

   ```
   $HOME/.kube/kubeconfig.cluster_name.vm
   ```

   Replace *cluster_name* with the name you used to create the cluster. The default is `ocne`.

   > **Tip**
   >
   > If you copy the file to the remote system as `$HOME/.kube/config`, you don't need to set the `$KUBECONFIG` environment variable to access the cluster on the remote host.

2. Install the `kubectl` package.

   ```
   sudo dnf install kubectl
   ```

3. Set the `kubeconfig` file location using an environment variable.

   For a cluster running on the localhost, use:

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

4. (Optional) Persist the environment variable.

   Add the environment variable to the `.bashrc` file. For example:

   ```
   echo 'export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)'
   >> $HOME/.bashrc
   ```

5. Verify that you can use `kubectl` to connect to the cluster.

   For example:

   ```
   kubectl get deployments --all-namespaces
   ```

Deleting a Cluster

Delete a Kubernetes cluster created using the `libvirt` provider.

Use the `ocne cluster delete` command to delete the cluster. They syntax to use is:

```
ocne cluster delete
[{-C|--cluster-name} name]
[{-c|--config} URI]
[{-P|--provider} provider]
[{-s|--session} URI]
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

Example 5-5 Delete the default `libvirt` cluster

To delete the default `libvirt` cluster:

```
ocne cluster delete
```

Example 5-6 Delete a cluster using the cluster name

To delete a cluster named `mycluster`:

```
ocne cluster delete --cluster-name mycluster
```

Example 5-7 Delete a cluster using a configuration file

To delete a cluster created using a configuration file:

```
ocne cluster delete --config myconfig.yaml
```

6

Oracle Linux Virtualization Manager Provider

Use the `olvm` provider to create Kubernetes clusters on Oracle Linux Virtualization Manager.

Kubernetes clusters are deployed to Oracle Linux Virtualization Manager using the `olvm` provider. The `olvm` provider is an implementation of the Kubernetes Cluster API. For information about the Kubernetes Cluster API, see the upstream documentation. The `olvm` provider uses the oVirt REST API to communicate with Oracle Linux Virtualization Manager. For information on the oVirt REST API, see the oVirt REST API upstream documentation.

Creating a cluster on Oracle Linux Virtualization Manager requires you to provide the OAuth 2.0 credentials to an existing instance of Oracle Linux Virtualization Manager. For information on OAuth 2.0, see the oVirt upstream documentation.

The `olvm` provider implements an infrastructure Cluster controller (OLVMCluster) Custom Resource Definition (CRD), and an infrastructure Machine controller (OLVMMachine) CRD, as Cluster API controllers. For each cluster, one OLVMCluster Custom Resource (CR) is needed, and an OLVMMachine CR for control plane nodes, and another for worker nodes. The configuration to create these CRs is set in a cluster configuration file.

The controllers that implement the Kubernetes Cluster API run inside a Kubernetes cluster. These clusters are known as *management clusters*. Management clusters control the life cycle of other clusters, known as *workload clusters*. A workload cluster can be its own management cluster.

Using the Kubernetes Cluster API to deploy a cluster on Oracle Linux Virtualization Manager requires that a Kubernetes cluster is available to use as the management cluster. Any running cluster can be used. To set the management cluster, set the `KUBECONFIG` environment variable, or use the `--kubeconfig` option of `ocne` commands. You could also set this cluster using a configuration file. If no management cluster is available, a cluster is created automatically using the `libvirt` provider, with the default configuration.

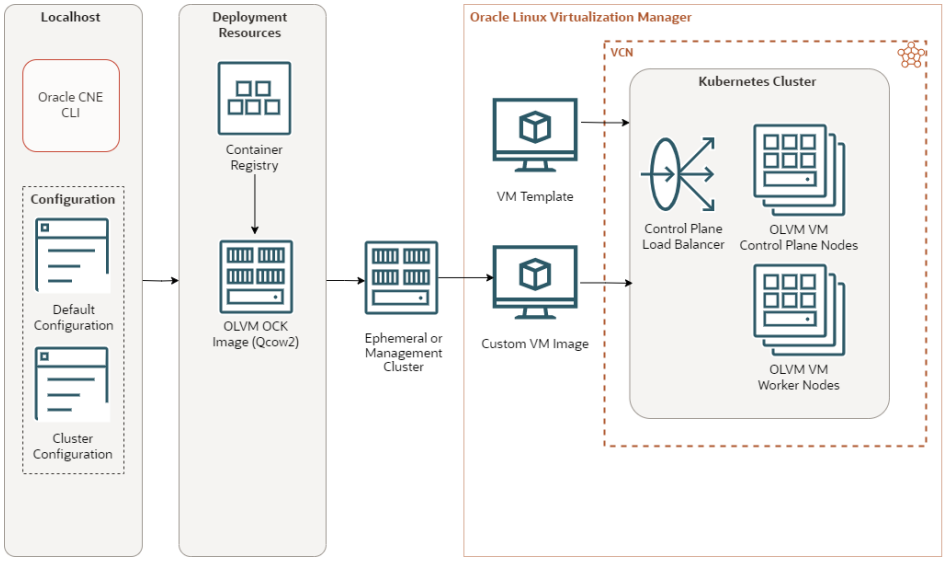
When a workload cluster has been deployed, it's managed using the Cluster API resources in the management cluster.

A workload cluster can be its own management cluster. This is known as a *self-managed cluster*. In a self-managed cluster, when the workload cluster has been deployed, the Cluster API resources are migrated into the workload cluster.

The `olvm` provider can be deployed using IPv4 IP addresses, or as a dual stack configuration using both IPv4 and IPv6 IP addresses. IPv6 on its own can't be used.

> **Note**
>
> Dynamic Host Configuration Protocol (DHCP) can't be used to assign IP addresses to the VMs used for Kubernetes nodes. Instead, provide a range of IP addresses for VMs to use as Kubernetes nodes, and set an IP address for the built-in Keepalived and NGINX load balancer for control plane nodes (the virtual IP).

Figure 6-1 Oracle Linux Virtualization Manager Cluster



The Oracle Linux Virtualization Manager cluster architecture has the following components:

- **CLI**: The CLI used to create and manage Kubernetes clusters. The `ocne` command.
- **Default configuration**: A YAML file that contains configuration for all `ocne` commands.
- **Cluster configuration**: A YAML file that contains configuration for a specific Kubernetes cluster.
- **Container registry**: A container registry used to pull the images used to create nodes in a Kubernetes cluster. The default is the Oracle Container Registry.
- **Oracle Linux Virtualization Manager OCK image**: The CLI is used to create this image, based on the OCK image, pulled from the container registry. The CLI is then used to upload this image to Oracle Linux Virtualization Manager.
- **Ephemeral or management cluster**: A Kubernetes cluster used to perform a CLI command. This cluster might also be used to start the cluster services, or to manage the cluster.
- **VM template**: An Oracle Linux Virtualization Manager VM template, used to create VMs in the Kubernetes cluster.
- **Custom VM image**: The OCK image is loaded into Oracle Linux Virtualization Manager as a custom VM image.
- **Control plane load balancer**: A network load balancer used for High Availability (HA) of the control plane nodes.
- **Control plane nodes**: VMs running control plane nodes in a Kubernetes cluster.
- **Worker nodes**: VMs running worker nodes in a Kubernetes cluster.

oVirt Container Storage Interface Driver

Describes the oVirt Container Storage Interface Driver (CSI) included with the Oracle Linux Virtualization Manager provider.

The oVirt CSI driver is installed into an Oracle Linux Virtualization Manager cluster by default, and automatically creates the required Kubernetes Secret, ConfigMap, and CsiDriver objects. The oVirt CSI driver lets you create persistent storage for Kubernetes applications using PersistentVolumes, which are backed by Oracle Linux Virtualization Manager storage disks.

To create an application that uses persistent storage, create a StorageClass, and a PersistentVolumeClaim, then set the application to use the PersistentVolumeClaim. When you start the application, a PersistentVolume is created on an Oracle Linux Virtualization Manager storage disk. Ensure you set the following in a StorageClass:

- `provisioner: csi.ovirt.org`
- `storageDomainName`: Set this to the name of the Oracle Linux Virtualization Manager storage domain. This is where PersistentVolumes are created.
- `thinProvisioning`: Set this to either `true` or `false`.
- `fsType`: Set the file system type, for example `ext4`. The driver formats the volume if no file system exists on the disk.

An example that uses this driver is in the Oracle CNE upstream documentation. For more information on creating Kubernetes storage objects, see the Kubernetes upstream documentation.

The default driver configuration is suitable for most deployments, however configuration options are available that can be set in a cluster configuration file. For information on the driver options, see the configuration file options in Oracle Linux Virtualization Manager Provider Options.

For more information on the oVirtCSI driver, see the Oracle CNE upstream documentation.

Setting Up the Oracle Linux Virtualization Manager Provider

Set up a system to create a Kubernetes cluster on Oracle Linux Virtualization Manager using the `olvm` provider.

To create a Kubernetes cluster on Oracle Linux Virtualization Manager, you need to set up the credentials to connect to an existing instance of Oracle Linux Virtualization Manager. You must also provide the location of a `kubeconfig` file for an existing Kubernetes cluster, or prepare the localhost to create an ephemeral cluster using the `libvirt` provider. The ephemeral cluster starts the Kubernetes Cluster API services used to create a cluster on Oracle Linux Virtualization Manager.

1. Set environment variables.

   The following environment variables must be set to provide OAuth 2.0 connection information about the Oracle Linux Virtualization Manager instance:

   - `OCNE_OLVM_USERNAME`: The username. The username must have `@internal` suffix. For example, if you sign in to the Oracle Linux Virtualization Manager console using the `admin` user, set this to `admin@internal`.
   - `OCNE_OLVM_PASSWORD`: The password for the user.
   - `OCNE_OLVM_SCOPE`: The authorization scope. Set this to `ovirt-app-api`, unless the user has a non default scope.

   For more information about OAuth 2.0, see the OAuth upstream documentation.

For example:

```
export OCNE_OLVM_USERNAME=admin@internal
export OCNE_OLVM_PASSWORD=password
export OCNE_OLVM_SCOPE=ovirt-app-api
```

2. Obtain the CA Certificate.

   You need to provide the CA Certificate to generate a key pair to connect to Oracle Linux Virtualization Manager. Save the CA Certificate on the localhost. Ensure you only use the second certificate returned by the instructions at oVirt CA.

   For example, to use `wget` to pull the CA Certificate in a file named `ca.crt`:

   ```
   wget --output-document ca.crt https://fqdn/ovirt-engine/services/pki-resource?resource=ca-certificate&format=X509-PEM-CA
   ```

   Replace *fqdn* with the fully qualified domain name (FQDN) of the Oracle Linux Virtualization Manager instance.

3. (Optional) Set the location of the `kubeconfig` file for an existing cluster.

   A Kubernetes cluster is required to perform some steps. You can use an existing cluster for this purpose by setting the location of the `kubeconfig` file.

   You can set this using the `KUBECONFIG` environment variable, or using the `--kubeconfig` option with `ocne` commands. You could also set this in a configuration file.

   If you don't set the location of the `kubeconfig` file, an ephemeral cluster is created using the `libvirt` provider when required.

4. (Optional) Set up the `libvirt` provider.

   If you don't set the location of an existing Kubernetes cluster, set up the localhost to provision ephemeral clusters using the `libvirt` provider. For information on setting up the `libvirt` provider, see Setting Up the libvirt Provider.

Creating a Configuration File for an Oracle Linux Virtualization Manager Cluster

Create a cluster configuration file for an Oracle Linux Virtualization Manager cluster.

Some mandatory configuration options need to be set in a configuration file before you create a cluster on Oracle Linux Virtualization Manager.

A cluster configuration contains the cluster specific information to use when creating the cluster. This file overrides any default configuration (set in the `$HOME/.ocne/defaults.yaml` file).

> **Note**
>
> You can include the cluster configuration information in the default configuration file. If you do this, you don't need to include the location of the cluster configuration file when you create the cluster using the `ocne cluster start` command.

Set up a cluster configuration file. A cluster configuration file must include at least the following options:

```
provider: olvm
name: cluster_name
virtualIp: IP_address
providers:
  olvm:
    olvmDatacenterName: datacenter_name
    olvmOvirtAPIServer:
      serverURL: URL
      serverCAPath: path
    olvmOCK:
      storageDomainName: domain_name
      diskName: image_name
      diskSize: disk_size
    controlPlaneMachine:
      olvmOvirtClusterName: cluster_name
      vmTemplateName: template_name
      olvmNetwork:
        networkName: network_name
        vnicProfileName: profile_name
      virtualMachine:
        memory: size
        network:
          gateway: IP_address
          ipv4:
            subnet: subnet
            ipAddresses: IPs
    workerMachine:
      olvmOvirtClusterName: cluster_name
      vmTemplateName: template_name
      olvmNetwork:
        networkName: network_name
        vnicProfileName: profile_name
      virtualMachine:
        memory: size
        network:
          gateway: IP_address
          ipv4:
            subnet: subnet
            ipAddresses: IPs
```

For information on what can be included in the cluster configuration file, see Oracle Linux Virtualization Manager Provider Options.

For example:

```
provider: olvm
name: mycluster
virtualIp: 192.0.2.21
providers:
  olvm:
    olvmDatacenterName: Default
    olvmOvirtAPIServer:
      serverURL: https://olvm.example.com/ovirt-engine
```

```
    serverCAPath: /home/username/olvm/ca.crt
  olvmOCK:
    storageDomainName: mydomain
    diskName: ock-1.33
    diskSize: 16GB
  controlPlaneMachine:
    olvmOvirtClusterName: myolvmcluster
    vmTemplateName: ock-1.33
    olvmNetwork:
      networkName: mynetwork
      vnicProfileName: myprofile
    virtualMachine:
      memory: 8GB
      network:
        gateway: 2.3.4.1
        ipv4:
          subnet: 192.0.0.0/24
          ipAddresses: 192.0.2.0/24, 192.0.2.10-192.0.2.20
  workerMachine:
    olvmOvirtClusterName: myolvmcluster
    vmTemplateName: ock-1.33
    olvmNetwork:
      networkName: mynetwork
      vnicProfileName: myprofile
    virtualMachine:
      memory: 16GB
      network:
        gateway: 2.3.4.1
        ipv4:
          subnet: 192.0.0.0/24
          ipAddresses: 192.0.2.0/24, 192.0.2.21-192.0.2.30, 192.0.2.32
```

A more complex cluster configuration file might include:

```
provider: olvm
proxy:
  httpsProxy: http://myproxy.example.com:2138
  httpProxy: http://myproxy.example.com:2138

noProxy: .example.com,127.0.0.1,localhost,169.254.169.254,10.96.0.0/12,10.244.
0.0/16
headless: true
name: mycluster
workerNodes: 3
controlPlaneNodes: 3
virtualIp: 192.0.2.21
providers:
  olvm:
    olvmDatacenterName: Default
    olvmOvirtAPIServer:
      serverURL: https://olvm.example.com/ovirt-engine
      serverCAPath: /home/username/olvm/ca.crt
      credentialsSecret:
        name: my-olvm-creds
        namespace: olvm
```

```
    caConfigMap:
      name: my-olvm-ca
      namespace: olvm
  olvmOCK:
    storageDomainName: mydomain
    diskName: ock-1.33
    diskSize: 16GB
  controlPlaneMachine:
    olvmOvirtClusterName: myolvmcluster
    vmTemplateName: ock-1.33
    olvmNetwork:
      networkName: mynetwork
      vnicName: nic-1
      vnicProfileName: myprofile
    virtualMachine:
      memory: 8GB
      network:
        gateway: 2.3.4.1
        ipv4:
          subnet: 192.0.0.0/24
          ipAddresses: 192.0.2.0/24, 192.0.2.10-192.0.2.20
  workerMachine:
    olvmOvirtClusterName: myolvmcluster
    vmTemplateName: ock-1.33
    olvmNetwork:
      networkName: mynetwork
      vnicName: nic-1
      vnicProfileName: myprofile
    virtualMachine:
      memory: 16GB
      network:
        gateway: 2.3.4.1
        ipv4:
          subnet: 192.0.0.0/24
          ipAddresses: 192.0.2.0/24, 192.0.2.21-192.0.2.30, 192.0.2.32
```

Creating an OCK Image for the Oracle Linux Virtualization Manager Provider

Create an Oracle Container Host for Kubernetes (OCK) image for the Oracle Linux Virtualization Manager (`olvm`) provider. Then upload the image to Oracle Linux Virtualization Manager so it an be used as the boot disk for VM instances.

1. Set up the Oracle Linux Virtualization Manager provider.

   For information on setting up the provider, see Setting Up the Oracle Linux Virtualization Manager Provider.

2. Create a configuration file.

   For information on creating a configuration file, see Creating a Configuration File for an Oracle Linux Virtualization Manager Cluster.

3. (Optional) Set the location of the `kubeconfig` file for an existing cluster.

A Kubernetes cluster is required to perform some steps. You can use an existing cluster for this purpose by setting the location of the `kubeconfig` file.

You can set this using the `KUBECONFIG` environment variable, or using the `--kubeconfig` option with `ocne` commands. You could also set this in a configuration file.

If you don't set the location of the `kubeconfig` file, an ephemeral cluster is created using the `libvirt` provider when required.

4. Create an OCK image.

   Use the `ocne image create` command to create an OCK image for Oracle Linux Virtualization Manager. The syntax is:

   ```
   ocne image create
   {-a|--arch} arch
   [{-t|--type} provider]
   [{-v|--version} version]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne image create --type olvm --arch amd64
   ```

   The Kubernetes cluster is used to generate the Qcow2 image, and the image is saved to the `$HOME/.ocne/images/` directory.

5. Upload the OCK image to Oracle Linux Virtualization Manager.

   Use the `ocne image upload` command to upload the image to Oracle Linux Virtualization Manager. The syntax is:

   ```
   ocne image upload
   {-a|--arch} arch
   [{-b|--bucket} name]
   [{-c|--compartment} name]
   [--config path]
   [{-d|--destination} path]
   {-f|--file} path
   {-i|--image-name} name
   {-t|--type} provider
   {-v|--version} version
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne image upload --type olvm --file $HOME/.ocne/images/boot.qcow2-1.33-
   amd64.olvm --arch amd64 --config myconfig.yaml
   ```

> **Note**
>
> A Kubernetes cluster is required for this step. If you don't have a Kubernetes cluster set, include the `--kubeconfig` option. For example, to set this to an ephemeral cluster you might use:
>
> ```
> ocne image upload --kubeconfig $HOME/.kube/kubeconfig.ocne.local ...
> ```

The image is uploaded and can now be used to create Oracle Linux Virtualization Manager VM instances to use in a Kubernetes cluster.

Creating an Oracle Linux Virtualization Manager VM Template

Use the Oracle Linux Virtualization Manager console to create a VM template that uses the OCK image uploaded to Oracle Linux Virtualization Manager. This template is used to create nodes in a Kubernetes cluster.

Define a VM template in Oracle Linux Virtualization Manager. This template is used to set the configuration for VM nodes in the cluster, such as CPUs, and memory size. This template is used to create the nodes in the cluster. You can create a template for control plane nodes, and another for worker nodes.

> **Tip**
>
> If you don't see the OS name `Oracle CNE - OCK x64` in Oracle Linux Virtualization Manager, upgrade the product branding package. This package must be `olvm-branding-4.5.5-1.3.el8.noarch.rpm` or later. Upgrade the package on the Oracle Linux Virtualization Manager host using:
>
> ```
> sudo dnf upgrade olvm-branding
> ```

1. Sign in to Oracle Linux Virtualization Manager.
2. Create a VM.

   Navigate to the **Virtual Machines** page, and create a VM.

   Fill in the form, only changing the following fields:

   Operating System: `Oracle CNE - OCK x64`

   Name: Set a name for the VM.

   Instance Image: Attach and select the uploaded `boot.qcow2` image, and select the **OS (boot)** checkbox.

   Select **OK** to create the VM.
3. Create a VM template.

   After the VM creation is finished, select the VM, and select **Make Template**. Set the name of the VM template to the value assigned to the `vmTemplateName` option in the cluster configuration file.

> **Note**
>
> The VM template name must match the `vmTemplateName` in the cluster configuration file.

4. Delete the VM.

   Delete the VM used to create the VM template. This is no longer required.

> **Note**
>
> Don't remove the VMs disk when you delete the VM. The disk is used by the VM template to create more VMs.

Creating an Oracle Linux Virtualization Manager Cluster

Create a Kubernetes cluster on Oracle Linux Virtualization Manager using the `olvm` provider.

Creating a Kubernetes cluster on Oracle Linux Virtualization Manager using the `olvm` provider requires that you first set up the localhost to create a management cluster using the `libvirt` provider, or have a cluster available to use as the management cluster. For information on the management cluster, see Oracle Linux Virtualization Manager Provider.

> **Caution**
>
> The number of control plane nodes must be an odd number (1, 3, 5, and so on) to avoid split brain scenarios with High Availability. The default is 1 control plane node and 0 worker nodes.

Create the cluster. Use the `ocne cluster start` command to create a cluster. The syntax is:

```
ocne cluster start
[{-u|--auto-start-ui} {true|false}]
[{-o|--boot-volume-container-image} URI]
[{-C|--cluster-name} name]
[{-c|--config} path]
[{-n|--control-plane-nodes} integer]
[{-i|--key} path]
[--load-balancer address]
[{-P|--provider} provider]
[{-s|--session} URI]
[{-v|--version} version]
[--virtual-ip IP]
[{-w|--worker-nodes} integer]
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

To use a cluster configuration file:

```
ocne cluster start --config myconfig.yaml
```

To use the settings in the default configuration file:

```
ocne cluster start --provider olvm
```

> **Tip**
>
> You can see the nodes as they're created using the Oracle Linux Virtualization Manager console. The control plane VMs are created first, then the worker node VMs.

Monitoring a Cluster Installation

View the logs for the Kubernetes Cluster API pods to monitor the creation of a Kubernetes cluster on Oracle Linux Virtualization Manager.

You can monitor the deployment of a Kubernetes cluster on Oracle Linux Virtualization Manager by reviewing the logs of several Kubernetes Cluster API pods that are created in the management cluster. You can also view information about Kubernetes machine objects created by the Kubernetes Cluster API.

> **Tip**
>
> If you set the cluster to be self managed, the ephemeral cluster is deleted after the deployment succeeds. To view the logs after the ephemeral cluster is deleted, set the cluster's `kubeconfig` file to the workload cluster running on Oracle Linux Virtualization Manager, for example:
>
> ```
> export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
> ```
>
> Where *cluster_name* is the name of the workload cluster.

1. Set the location of the `kubeconfig` file for the management cluster.

   In a separate terminal session, set the location of the management cluster's `kubeconfig` file.

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Where *cluster_name* is the name of the management cluster.

2. View the events.

   Use the `kubectl get events` command to get information about the events in the namespace in which the cluster is created. The default namespace is `olvm`. For example:

   ```
   kubectl get events --namespace olvm
   ```

3. View the Machine objects.

   Use the `kubectl get machine` command to get information about the Kubernetes Machine objects.

   ```
   kubectl get machine --namespace olvm
   ```

   The `NAME` of each VM matches the VM name in Oracle Linux Virtualization Manager. If the nodes are created successfully, the `PHASE` column is `Running` for each Machine.

   To get more information about each Machine, use the `kubectl describe` command, for example:

   ```
   kubectl describe machine --namespace olvm cluster_name-control-plane-ID
   ```

4. View the OLVMMachine objects.

   Use the `kubectl get olvmmachine` command to get information about the Kubernetes OLVMMachine objects.

   ```
   kubectl get olvmmachine --namespace olvm
   ```

   Again, the `NAME` of each VM matches the VM name in Oracle Linux Virtualization Manager. If the nodes are created successfully, the `READY` column is `true` for each Machine.

   To get more information about each OLVMMachine, use the `kubectl describe` command, for example:

   ```
   kubectl describe olvmmachine --namespace olvm cluster_name-control-plane-ID
   ```

5. View the `core-capi-controller-manager` pod logs.

   Use the `kubectl logs` command to view the logs for the pod.

   Copy the command listed and press the `Tab` key to access the full pod name.

   ```
   kubectl logs --namespace capi-system core-capi-controller-manager
   ```

6. View the `olvm-capi-controller-manager` pod logs.

   Use the `kubectl logs` command to view the logs for the pod.

   Copy the command listed and press the `Tab` key to access the full pod name.

   ```
   kubectl logs --namespace cluster-api-provider-olvm olvm-capi-controller-
   manager
   ```

7. View the `control-plane-capi-controller-manager` pod logs.

   Use the `kubectl logs` command to view the logs for the pod.

   Copy the command listed and press the `Tab` key to access the full pod name.

   ```
   kubectl logs --namespace capi-kubeadm-control-plane-system control-plane-
   capi-controller-manager
   ```

8. View the `bootstrap-capi-controller-manager` pod logs.

   Use the `kubectl logs` command to view the logs for the pod.

Copy the command listed and press the `Tab` key to access the full pod name.

```
kubectl logs --namespace capi-kubeadm-bootstrap-system bootstrap-capi-
controller-manager
```

Connecting to a Cluster

Use the `kubectl` package to connect to a Kubernetes cluster.

After creating a Kubernetes cluster, a Kubernetes configuration file is created so you can access the cluster using the `kubectl` command. Install `kubectl` on the localhost (the host with `ocne` installed on which you created the cluster). If you install `kubectl` on a different system, copy the `kubeconfig` file to the system.

1. Install the `kubectl` package on the localhost.

   ```
   sudo dnf install kubectl
   ```

2. Set the `kubeconfig` file location using an environment variable.

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Replace *cluster_name* with the name you used to create the cluster. The default is `ocne`.

3. (Optional) Persist the environment variable.

   Add the environment variable to the `.bashrc` file:

   ```
   echo 'export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)'
   >> $HOME/.bashrc
   ```

4. Verify that you can use `kubectl` to connect to the cluster.

   For example:

   ```
   kubectl get deployments --all-namespaces
   ```

Scale a Cluster

Learn how to scale a Kubernetes cluster on Oracle Linux Virtualization Manager.

You can scale the cluster nodes by using the `kubectl scale` command, either on the management cluster, or directly in the Oracle Linux Virtualization Manager workload cluster if it's self-managed.

> **Note**
>
> You can also scale cluster nodes by editing the `replicas` option in the appropriate Kubernetes Cluster API Custom Resource file.

Scaling Worker Nodes in an Oracle Linux Virtualization Manager Cluster

Use the `kubectl scale` command to scale the number of worker nodes in a Kubernetes cluster on Oracle Linux Virtualization Manager.

You can scale the worker nodes in a Kubernetes cluster by running the `kubectl scale` command, either on the management cluster, or on the Oracle Linux Virtualization Manager workload cluster if it's self-managed.

1. Get the name of the `machinedeployments` Custom Resource.

   Use the `kubectl get` command to retrieve this information, for example:

   ```
   kubectl get machinedeployments --namespace olvm
   ```

2. Run the `kubectl scale` command, specifying the new number of worker nodes.

   Set the new number of worker nodes with the `--replicas` option. In this example, the name of the `machinedeployments` Custom Resource is `mycluster-md-0`:

   ```
   kubectl scale machinedeployment mycluster-md-0 --replicas 3 --namespace
   olvm
   ```

3. Wait for the cluster to stabilize before performing any other scaling operations.

   You can monitor the status of the nodes by running the following command on the Oracle Linux Virtualization Manager cluster:

   ```
   watch kubectl get nodes -A
   ```

   The cluster is stable when all the cluster nodes show a status of `Ready`. Exit the command using Ctrl+C.

   The number of worker nodes in the cluster is scaled up or down to the number of replicas you set.

Scaling Control Plane Nodes in an Oracle Linux Virtualization Manager Cluster

Use the `kubectl scale` command to scale the number of control plane nodes in a Kubernetes cluster on Oracle Linux Virtualization Manager.

You can scale the control plane nodes in a Kubernetes cluster by running the `kubectl scale` command, either on the management cluster, or on the Oracle Linux Virtualization Manager workload cluster if it's self-managed.

1. Get the name of the `kubeadmcontrolplane` Custom Resource.

   Use the `kubectl get` command to retrieve this information, for example:

   ```
   kubectl get kubeadmcontrolplane --namespace olvm
   ```

2. Run the `kubectl scale` command, specifying the new number of control plane nodes.

Set the new number of control plane nodes with the `--replicas` option. In this example, the name of the `kubeadmcontrolplane` Custom Resource is `mycluster-control-plane`:

```
kubectl scale kubeadmcontrolplane mycluster-control-plane --replicas 3 --
namespace olvm
```

3. Wait for the cluster to stabilize before performing any other scaling operations.

   You can monitor the status of the nodes by running the following command on the Oracle Linux Virtualization Manager cluster:

   ```
   watch kubectl get nodes -A
   ```

   The cluster is stable when all the cluster nodes show a status of `Ready`. Exit the command using Ctrl+C.

   The number of control plane nodes in the cluster is scaled up or down to the number of replicas you set.

Upgrading an Oracle Linux Virtualization Manager Cluster to a Kubernetes Minor Release

Upgrade an Oracle Linux Virtualization Manager cluster to the next Kubernetes minor release.

For clusters running on Oracle Linux Virtualization Manager, and created using the `olvm` provider, upgrade a Kubernetes cluster to the next minor Kubernetes version when an OCK image becomes available for that version. This uses the Kubernetes Cluster API to scale nodes in and out. To perform an in place upgrade, where nodes aren't reprovisioned, use the steps in Upgrading to a Kubernetes Minor Release.

1. Create an OCK image.

   Create and upload a new OCK image that contains the updated Kubernetes version. For information creating an OCK image, see Creating an OCK Image for the Oracle Linux Virtualization Manager Provider.

2. Create a VM template.

   Create a VM template that uses the new OCK image. This is used to create new VMs with the updated version of Kubernetes. For information on creating a VM template, see Creating an Oracle Linux Virtualization Manager VM Template.

3. Edit the cluster configuration file.

   Edit the cluster configuration file used to create the Kubernetes cluster. Change the `vmTemplateName` entries to match the new VM template name. Change this entry in both the `controlPlaneMachine` and `workerMachine` sections. For example:

   ```
   providers:
     olvm:
   ...
       controlPlaneMachine:
         vmTemplateName: ock-1.33
   ...
       workerMachine:
         vmTemplateName: ock-1.33
   ...
   ```

4. Set the location of the management cluster.

   The management cluster contains the Kubernetes Cluster API controllers. The management cluster might be the same as the workload cluster (a self-managed cluster). The upgrade is performed using the management cluster.

   Set the `kubeconfig` file location to the management cluster using an environment variable.

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Replace *cluster_name* with the name of the management cluster.

5. Set the target Kubernetes version.

   Use `ocne cluster stage` command to stage the target Kubernetes version. Use the configuration file used to create the workload cluster with this command. The syntax to use is:

   ```
   ocne cluster stage
   [{-c|--config} path]
   [{-r|--os-registry} registry]
   [{-t|--transport} transport]
   {-v|--version} version
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster stage --version 1.33 --config mycluster.yaml
   ```

   The output of this command prints important information. For example, the output might look similar to:

   ```
   To update KubeadmControlPlane ocne-control-plane in olvm-cluster, run:
       kubectl patch -n olvm-cluster kubeadmcontrolplane ocne-control-plane --
   type=json -p='[{"op":"replace","path":"/spec/version","value":"1.33.0"},
   {"op":"replace","path":"/spec/machineTemplate/infrastructureRef/
   name","value":"ocne-control-plane-1"}]'

   To update MachineDeployment ocne-md-0 in olvm-cluster, run:
       kubectl patch -n olvm-cluster machinedeployment ocne-md-0 --type=json -
   p='[{"op":"replace","path":"/spec/template/spec/version","value":"1.33.0"},
   {"op":"replace","path":"/spec/template/spec/infrastructureRef/
   name","value":"ocne-md-1"}]'
   ```

6. Patch the KubeadmControlPlane for control plane nodes.

   Use the `kubectl patch` command to update the KubeadmControlPlane. Use the command printed in the output of the `ocne cluster stage` command. For example:

   ```
    kubectl patch -n olvm-cluster kubeadmcontrolplane ocne-control-plane --
   type=json -p='[{"op":"replace","path":"/spec/version","value":"1.33.0"},
   {"op":"replace","path":"/spec/machineTemplate/infrastructureRef/
   name","value":"ocne-control-plane-1"}]'
   ```

The control plane nodes are reprovisioned using the new OCK image that includes the new version of Kubernetes. This might take some time.

> **Tip**
>
> Monitor new nodes are being provisioned, and old nodes are being removed, using:
>
> ```
> kubectl --namespace namespace get machine
> ```
>
> You can also monitor the VMs are being created and destroyed using the Oracle Linux Virtualization Manager console.

7. Confirm control plane nodes are upgraded.

   Confirm all nodes are upgraded in the workload cluster. In a separate terminal, set the `kubeconfig` file location to the workload cluster using an environment variable.

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Replace *cluster_name* with the name of the workload cluster.

   List the nodes in the cluster.

   ```
   kubectl get nodes
   ```

   Confirm the `VERSION` column lists the new Kubernetes version number.

8. Update the MachineDeployment for worker nodes.

   In the terminal where the `kubeconfig` is set to the management cluster, use the `kubectl patch` command to update the MachineDeployment for worker nodes. Use the command printed in the output of the `ocne cluster stage` command. For example:

   ```
   kubectl patch -n olvm-cluster machinedeployment ocne-md-0 --type=json -
   p='[{"op":"replace","path":"/spec/template/spec/version","value":"1.33.0"},
   {"op":"replace","path":"/spec/template/spec/infrastructureRef/
   name","value":"ocne-md-1"}]'
   ```

   The worker nodes are reprovisioned using the new OCK image, with the new version of Kubernetes. This might take some time.

9. Confirm worker nodes are upgraded.

   Confirm all worker nodes are upgraded in the workload cluster. In the separate terminal, where the `kubeconfig` is set to the workload cluster, list the nodes in the cluster.

   ```
   kubectl get nodes
   ```

   Confirm the `VERSION` column lists the new Kubernetes version number.

Delete a Cluster

Learn how to delete a Kubernetes cluster on Oracle Linux Virtualization Manager.

Deleting a self-managed cluster requires that a second cluster is available to run the Kubernetes Cluster API controllers (a management cluster). This is because the final stages of cluster destruction destroys any remaining VMs, and, by extension, cluster nodes. When this is complete, no VMs are available to run the controllers. If a management cluster isn't available, an ephemeral cluster is created to service this need.

Deleting an Oracle Linux Virtualization Manager Cluster

1. Delete the Kubernetes cluster running on Oracle Linux Virtualization Manager.

   Use the `ocne cluster delete` command to delete the cluster. They syntax to use is:

   ```
   ocne cluster delete
   [{-C|--cluster-name} name]
   [{-c|--config} URI]
   [{-P|--provider} provider]
   [{-s|--session} URI]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster delete --cluster-name mycluster
   ```

   or:

   ```
   ocne cluster delete --config myconfig.yaml
   ```

   The management cluster deletes the Oracle Linux Virtualization Manager workload cluster. If the cluster is self managed, an ephemeral cluster is created and the Kubernetes Cluster API resources are migrated to the ephemeral cluster.

   Tip

   Monitor the cluster deletion process in a separate terminal session. Set the `KUBECONFIG` environment variable to the management cluster.

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Replace *cluster_name* with the name of the management cluster. Then enter:

   ```
   kubectl get cluster -A
   ```

   The cluster object is in the `deleting` state. When the cluster object is no longer available, the delete is completed.

2. Delete the management cluster.

You might also need to delete the management cluster, depending on the settings in the environment. Use the `ocne cluster delete` command to delete the management cluster. For example:

```
ocne cluster delete --cluster-name mycluster
```

7

OCI Provider

Use the `oci` provider to create Kubernetes clusters on Oracle Cloud Infrastructure (OCI).

Kubernetes clusters are deployed to OCI using the `oci` provider. The `oci` provider uses the Kubernetes Cluster API Provider for OCI to perform the deployment. This is an implementation of the Kubernetes Cluster API. The Kubernetes Cluster API is implemented as Kubernetes Custom Resources (CRs), that are serviced by applications running in a Kubernetes cluster. The Kubernetes Cluster API has a large interface and is explained in the upstream documentation. For information on the Kubernetes Cluster API, see the Kubernetes Cluster API documentation. For information on the Cluster API implementation for OCI, see the Kubernetes Cluster API Provider for OCI documentation.

Creating a cluster on OCI requires you to provide the credentials to an existing tenancy. The required privileges depend on the configuration of the cluster that's created. For some deployments, it might be enough to have the privileges to create and destroy compute instances. For other deployments, more privilege might be required.

Clusters are deployed into specific compartments. The `oci` provider requires that a compartment is available. Compartments can be specified either by the Oracle Cloud Identifier (OCID), or by its path in the compartment hierarchy, for example, `parentcompartment/mycompartment`.

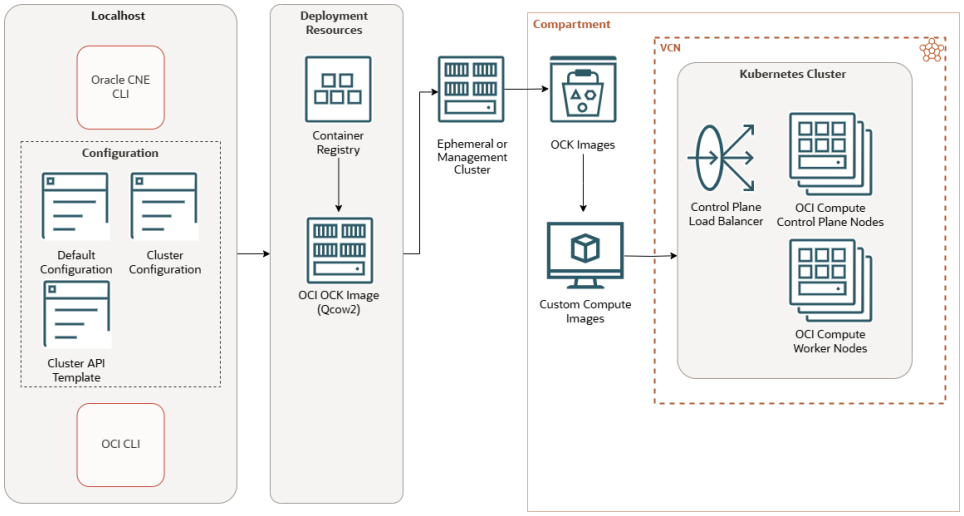
The controllers that implement the Kubernetes Cluster API run inside a Kubernetes cluster. These clusters are known as *management clusters*. Management clusters control the life cycle of other clusters, known as *workload clusters*. A workload cluster can be its own management cluster.

Using the Kubernetes Cluster API to deploy a cluster on OCI requires that a Kubernetes cluster is available to use as the management cluster. Any running cluster can be used. To set the management cluster, set the `KUBECONFIG` environment variable, or use the `--kubeconfig` option of `ocne` commands. You could also set this cluster using a configuration file. If no management cluster is available, a cluster is created automatically using the `libvirt` provider, with the default configuration.

When a workload cluster has been deployed, it's managed using the Kubernetes Cluster API resources in the management cluster.

A workload cluster can be its own management cluster. This is known as a *self-managed cluster*. In a self-managed cluster, when the workload cluster has been deployed, the Cluster API resources are migrated into the workload cluster.

Figure 7-1 OCI Cluster



The OCI cluster architecture has the following components:

- **CLI**: The CLI used to create and manage Kubernetes clusters. The `ocne` command.
- **Default configuration**: A YAML file that contains configuration for all `ocne` commands.
- **Cluster configuration**: A YAML file that contains configuration for a specific Kubernetes cluster.
- **Cluster API template**: A YAML file that contains Cluster Resources for the Kubernetes Cluster API to create a cluster.
- **OCI CLI**: The OCI CLI is installed on the localhost, including the configuration to read and write to the tenancy and compartment.
- **Container registry**: A container registry used to pull the images used to create nodes in a Kubernetes cluster. The default is the Oracle Container Registry.
- **OCI OCK image**: The CLI is used to create this image, based on the OCK image, pulled from the container registry. The CLI is then used to upload this image to OCI.
- **Ephemeral or management cluster**: A Kubernetes cluster used to perform a CLI command. This cluster might also be used to start the cluster services, or to manage the cluster.
- **Compartment**: An OCI compartment in which the cluster is created.
- **OCK images**: The OCK image is loaded into an Object Storage bucket. When the upload is complete, a custom compute image is created from the OCK image.
- **Custom compute images**: The OCK image is available as a custom compute image and can be used to create compute nodes in a Kubernetes cluster.
- **Control plane load balancer**: A network load balancer used for High Availability (HA) of the control plane nodes.
- **Control plane nodes**: Compute instances running control plane nodes in a Kubernetes cluster.
- **Worker nodes**: Compute instances running worker nodes in a Kubernetes cluster.

Setting Up the OCI Provider

Set up a system to create a Kubernetes cluster on OCI using the `oci` provider.

To create a Kubernetes cluster on OCI, you need to set up the OCI CLI and create an Object Storage bucket. You must also provide the location of a `kubeconfig` file for an existing Kubernetes cluster, or prepare the localhost to create an ephemeral cluster using the `libvirt` provider. The ephemeral cluster starts the Kubernetes Cluster API services used to create a cluster on OCI.

1. Set up the OCI CLI.

   Install and configure the OCI CLI. Ensure you set up the key pair and configuration file. For information on setting up the CLI, see the OCI documentation.

2. Create an Object Storage bucket.

   Create an Object Storage bucket in OCI named `ocne-images`.

   > **Tip**
   >
   > You can also specify the name of a bucket with another name in a configuration file.

   For information on creating an Object Storage bucket, see the Oracle Cloud Infrastructure documentation.

3. Set the required OCI configuration.

   Some information about the OCI environment is required before you can create a cluster. You need to set:

   - The compartment in which to deploy resources. This can be a path to a compartment, or the OCID.
   - (Optional) The Virtual Cloud Network (VCN) to use when provisioning the control plane load balancer.
   - (Optional) The subnets to use when provisioning the control plane load balancer.

   The VCN and subnet information is used by the OCI Cloud Controller Manager during deployment to configure the control plane load balancer. This configuration isn't used in the cluster itself, and doesn't impact the OCI resources deployed by the Kubernetes Cluster API controller and templates.

   If you don't set the control plane load balancer networking options, they're automatically set. You can set these options in a configuration file, or in a Kubernetes Cluster API template. If you set this information in the default configuration file (`$HOME/.ocne/defaults.yaml`), you might need to create the configuration file as it isn't created by default. You can also include these options in a cluster configuration file, and not in the default configuration file, however, you must include the cluster configuration file as a option when creating a cluster using the `ocne cluster start` command.

   The format to use is:

   ```
   providers:
     oci:
       compartment: OCID
   ```

```
    vcn: OCID
    loadBalancer:
      subnet1: OCID
      subnet2: OCID
```

For example:

```
providers:
  oci:
    compartment: ocid1.compartment.oc1..uniqueID
```

Or to include the networking configuration for the control plane node load balancer:

```
providers:
  oci:
    compartment: ocid1.compartment.oc1..uniqueID
    vcn: ocid1.vcn.oc1.uniqueID
    loadBalancer:
      subnet1: ocid1.subnet.oc1.uniqueID
      subnet2: ocid1.subnet.oc1.uniqueID
```

For more information on the default configuration file, see Oracle Cloud Native Environment: CLI.

4. (Optional) Set the location of the `kubeconfig` file for an existing cluster.

   A Kubernetes cluster is required to perform some steps. You can use an existing cluster for this purpose by setting the location of the `kubeconfig` file.

   You can set this using the `KUBECONFIG` environment variable, or using the `--kubeconfig` option with `ocne` commands. You could also set this in a configuration file.

   If you don't set the location of the `kubeconfig` file, an ephemeral cluster is created using the `libvirt` provider when required.

5. (Optional) Set up the `libvirt` provider.

   If you don't set the location of an existing Kubernetes cluster, set up the localhost to provision ephemeral clusters using the `libvirt` provider. For information on setting up the `libvirt` provider, see Setting Up the libvirt Provider.

6. (Optional) Create an Oracle Container Host for Kubernetes (OCK) image.

   If you're using customized Kubernetes Cluster API template files, first create an OCK image for the compute instance architecture, and upload it to OCI. For information on creating an OCK image, see Creating an OCK Image for the OCI Provider.

Cluster API Templates

Describes Kubernetes Cluster API template files in Oracle CNE.

A default OCI cluster can be created using the `oci` provider with no extra arguments. For example:

```
ocne cluster start --provider oci
```

The default cluster settings create a useful cluster, but, it's likely that extra configuration is required. To customize a deployment, generate a cluster template to use as a basis for the cluster.

The `ocne cluster template` command is used to create a cluster template, and uses the default configuration and any cluster configuration you set to generate the template. It also fetches things such as compute image OCIDs from the configured compartment automatically. You can create a cluster template and save it to a file, for example:

```
ocne cluster template > mytemplate.yaml
```

The resulting YAML file contains the Cluster Resources for the Kubernetes Cluster API to create a cluster, using all the configuration you have on the local system.

For example, you could create a cluster configuration file (named `myconfig.yaml` in this example) that includes customization to the deployment, and might include:

```
provider: oci
name: mycluster
controlPlaneNodes: 3
workerNodes: 3
clusterDefinition: /home/username/mytemplate.yaml
providers:
  oci:
    compartment: ocid1.compartment.oc1..uniqueID
```

In this example, the `clusterDefinition` file is the location of a Kubernetes Cluster API template, which is generated based on the configuration in this cluster configuration file. You use the template when you create the cluster. You can update the cluster template to use this new configuration by running the `ocne cluster template` command again and providing the configuration file. For example:

```
ocne cluster template --config myconfig.yaml > $HOME/mytemplate.yaml
```

Edit the template file to configure any Kubernetes Cluster API options that aren't provided in a cluster configuration file. When you're satisfied with the template, create the cluster using the cluster configuration file. As this cluster configuration file includes a link to the cluster template in the `clusterDefinition`, the template is used to create the cluster. For example:

```
ocne cluster start --config myconfig.yaml
```

Cluster API Template Files

Describes Kubernetes Cluster API template file contents.

A Kubernetes Cluster API template file can be generated that contains all the information required to create a Kubernetes cluster using the Kubernetes Cluster API. The `ocne cluster template` command is used to create this file. Save and edit this template to create clusters using the Kubernetes Cluster API.

Optionally, edit the Custom Resources in the template to suit the cluster you want to create. The options available are described in the upstream Kubernetes Cluster API Provider for Oracle Cloud Infrastructure documentation.

The template file contains Custom Resources for each component of a cluster, and includes:

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  labels:
    cluster.x-k8s.io/cluster-name: "ocne"
  name: "ocne"
  namespace: "ocne"
spec:
...
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta2
kind: OCICluster
metadata:
  labels:
    cluster.x-k8s.io/cluster-name: "ocne"
  name: "ocne"
  namespace: "ocne"
spec:
...
---
kind: KubeadmControlPlane
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
metadata:
  name: "ocne-control-plane"
  namespace: "ocne"
spec:
...
---
kind: OCIMachineTemplate
apiVersion: infrastructure.cluster.x-k8s.io/v1beta2
metadata:
  name: "ocne-control-plane"
  namespace: "ocne"
spec:
...
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta2
kind: OCIMachineTemplate
metadata:
  name: "ocne-md-0"
  namespace: "ocne"
spec:
...
---
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfigTemplate
metadata:
  name: "ocne-md-0"
  namespace: "ocne"
spec:
...
```

Creating a Cluster API Template

Create a Kubernetes Cluster API template using the `ocne cluster template` command.

A Kubernetes Cluster API template can be used when deploying clusters using the Kubernetes Cluster API.

You can generate a Cluster API template using the defaults set on the local system, or using a cluster configuration file.

1. Create a cluster template.

   Use the `ocne cluster template` command to generate a YAML file that contains a template to create a cluster using the Kubernetes Cluster API. The syntax is:

   ```
   ocne cluster template
   [{-c|--config} path]
   [{-P|--provider} provider]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster template --config myconfig.yaml > mytemplate.yaml
   ```

2. (Optional) Edit the template.

   Edit the file to suit the requirements of the cluster you want to create. The options available are described in the upstream Kubernetes Cluster API Provider for Oracle Cloud Infrastructure documentation.

Using an Existing VCN

Edit a Cluster API template to use an existing OCI Virtual Cloud Network (VCN).

To use an existing VCN when deploying a Kubernetes cluster to OCI, generate and edit a Cluster API template to include the VCN information. You need to provide OCIDs for:

- VCN. The VCN to use for the cluster nodes.
- Network Security Group. This is the network security group that contains the ingress rules to the VCN.
- Subnet. This is the subnet of the VCN.

The values provided for the VCN and subnet are also used by the OCI Cloud Controller Manager during deployment to configure the control plane load balancer.

1. Create a Cluster API template.

   Set up the configuration files to create the cluster, including all relevant OCIDs and generate a Cluster API template. For information on creating a Cluster API template, see Creating a Cluster API Template.

2. Set the OCIDs as environment variables.

Replace *OCID* in each line with the OCID for the network object.

```
export VCN=OCID
export SECGROUP=OCID
export SUBNET=OCID
```

3. Create a YAML file with the VCN information.

   Generate a file to include the VCN information, using the environment variables.

```
envsubst > vcn_config.yaml << EOF
  networkSpec:
    skipNetworkManagement: true
    vcn:
      id: $VCN
      networkSecurityGroup:
        skip: true
        list:
        - name: control-plane-endpoint
          role: control-plane-endpoint
          id: $SECGROUP
        - name: control-plane
          role: control-plane
          id: $SECGROUP
        - name: worker
          role: worker
          id: $SECGROUP
        - name: service-lb
          role: service-lb
          id: $SECGROUP
      internetGateway:
        skip: true
      natGateway:
        skip: true
      serviceGateway:
        skip: true
      routeTable:
        skip: true
      subnets:
      - name: control-plane-endpoint
        role: control-plane-endpoint
        id: $SUBNET
        type: private
      - name: control-plane
        role: control-plane
        id: $SUBNET
        type: private
      - name: worker
        role: worker
        id: $SUBNET
        type: private
      - name: service-lb
        role: service-lb
        id: $SUBNET
        type: private
EOF
```

4. Edit the Cluster API template.

   Open the file that includes the VCN information (`vcn_config.yaml` in this example), copy the content, and paste it into the Cluster API template. The information must be included in the `OCICluster` Custom Resource, in the `spec` section. For example:

   ```
   apiVersion: infrastructure.cluster.x-k8s.io/v1beta2
   kind: OCICluster
   metadata:
     labels:
       cluster.x-k8s.io/cluster-name: "mycluster"
     name: "mycluster"
     namespace: "ocne"
   spec:
     compartmentId: "ocid1.compartment.oc1..."
     # Paste the VCN information here.
     networkSpec:
       skipNetworkManagement: true
       vcn: ...
   ```

5. Create the cluster with the Cluster API template.

   Ensure you include the location of the Cluster API template in the cluster configuration file when you create the cluster. The configuration file must include the location of the template using the `clusterDefinition` option. For example:

   ```
   clusterDefinition: /home/username/mytemplate.yaml
   ```

OCI Compute Images

Describes OCK images used to create Kubernetes nodes on OCI.

Creating clusters with the OCI provider requires a custom compute image in the target compartment. The bootable container image must be customized to work in OCI, and must be converted into an appropriate format. After an appropriate image has been created, it must be imported to the target compartment.

> **Note**
>
> If you're using customized Kubernetes Cluster API template files, you first need to create an image for the compute instance architecture, and upload it to OCI. If you're not using customized templates, you don't need to create and upload the image manually, as it's done for you automatically when you create the Kubernetes cluster.

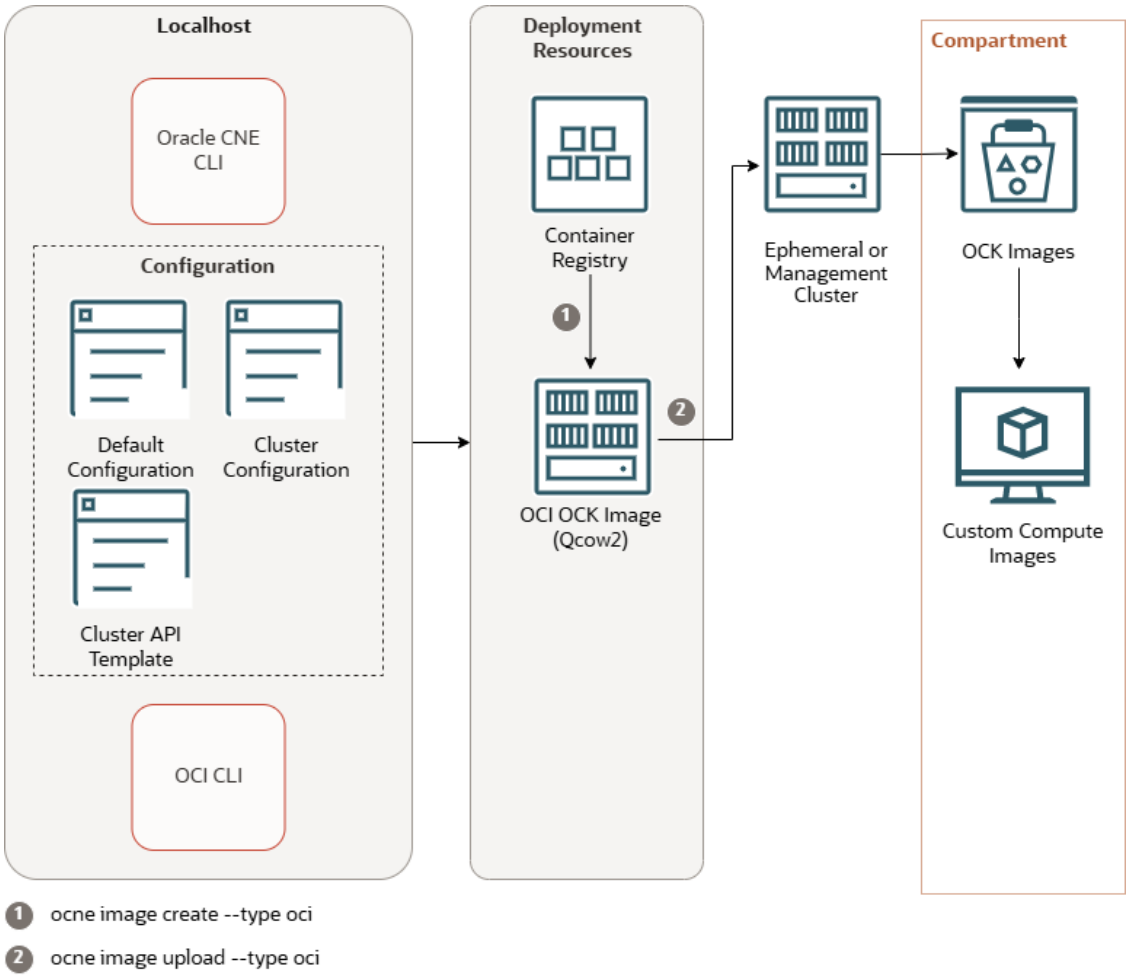
Bootable OCK images in Qcow2 format can be created using the `ocne image create` command with the `--type oci` option. By default, the image is created for the architecture of the system where the command is run. Images can be created for other architectures using the `--arch` option.

The resulting OCK image can be imported into OCI using the `ocne image upload` command, and used as the boot volume for compute instances. When you use the `ocne image upload` command to upload a Qcow2 image, the conversion to the appropriate format is performed automatically.

The image is uploaded to an OCI Object Storage bucket. After the upload is complete, the object is imported as a custom compute image.

The custom compute image can then be used to create compute instances for a Kubernetes cluster.

Figure 7-2 OCK Images for an OCI Cluster



The architecture has the following components:

- **CLI**: The CLI used to create and manage Kubernetes clusters. The `ocne` command.
- **Default configuration**: A YAML file that contains configuration for all `ocne` commands.
- **Cluster configuration**: A YAML file that contains configuration for a specific Kubernetes cluster.
- **Cluster API template**: A YAML file that contains Cluster Resources for the Kubernetes Cluster API to create a cluster.
- **OCI CLI**: The OCI CLI is installed on the localhost, including the configuration to read and write to the tenancy and compartment.
- **Container registry**: A container registry used to pull the images used to create nodes in a Kubernetes cluster. The default is the Oracle Container Registry.

- **OCI OCK image**: The CLI is used to create this image, based on the OCK image, pulled from the container registry. The CLI is then used to upload this image to OCI.
- **Ephemeral or management cluster**: A Kubernetes cluster used to perform a CLI command.
- **OCI Object Storage bucket**: The OCK image is loaded into an Object Storage bucket. When the upload is complete, a custom compute image is created from the OCK image.
- **Custom compute image**: The OCK image is available as a custom compute image and can be used to create compute nodes in a Kubernetes cluster.

The CLI uses any configuration files supplied to generate an OCK image. The CLI pulls the bootable Qcow2 OCK image from the container registry, and coverts it to an image that can be used for OCI. The CLI loads the OCK image to an OCI Object Storage bucket. The OCK image is then converted to a custom compute image, which can be used to create compute nodes in a Kubernetes cluster.

> **Note**
>
> The object in the bucket isn't automatically removed, and must be cleaned up manually when no longer required.

Creating an OCK Image for the OCI Provider

Create an Oracle Container Host for Kubernetes (OCK) image for the OCI (`oci`) provider. Then upload the image to OCI so it an be used as the boot disk for compute instances.

1. Set up the OCI provider.

   For information on setting up the provider, see Setting Up the OCI Provider.

2. (Optional) Set the location of the `kubeconfig` file for an existing cluster.

   A Kubernetes cluster is required to perform some steps. You can use an existing cluster for this purpose by setting the location of the `kubeconfig` file.

   You can set this using the `KUBECONFIG` environment variable, or using the `--kubeconfig` option with `ocne` commands. You could also set this in a configuration file.

   If you don't set the location of the `kubeconfig` file, an ephemeral cluster is created using the `libvirt` provider when required.

3. Create an OCK image.

   Use the `ocne image create` command to create an OCK image for OCI. The syntax is:

   ```
   ocne image create
   {-a|--arch} arch
   [{-t|--type} provider]
   [{-v|--version} version]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne image create --type oci --arch amd64
   ```

The Kubernetes cluster is used to generate the Qcow2 image, and the image is saved to the `$HOME/.ocne/images/` directory.

4. Upload the OCK image to OCI.

   Use the `ocne image upload` command to upload the image to OCI. The syntax is:

   ```
   ocne image upload
   {-a|--arch} arch
   [{-b|--bucket} name]
   [{-c|--compartment} name]
   [--config path]
   [{-d|--destination} path]
   {-f|--file} path
   {-i|--image-name} name
   {-t|--type} provider
   {-v|--version} version
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne image upload --compartment ocid1.compartment.oc1..UniqueID --
   file $HOME/.ocne/images/boot.qcow2-1.33-amd64.oci --arch amd64
   ```

   The Kubernetes cluster is used to upload the Qcow2 image. The image is uploaded to the Object Bucket store, and then automatically converted to a custom compute image. The image can now be used to create OCI instances to use in a Kubernetes cluster.

   Tip

   Sign in to the OCI console to monitor the upload of the image to the Object Bucket store, and the creation of a custom compute image.

Create a Cluster on OCI

Learn how to create a Kubernetes cluster on OCI.

When you create a cluster using the `oci` provider, the following occurs:

1. The CLI detects if a management cluster is available. If not, it starts a cluster to act as a management cluster. Any resources required to start the management cluster are fetched and installed.
2. When the management cluster is available, the configured OCI compartment is checked for compatible compute Oracle Container Host for Kubernetes (OCK) images. If no OCK images are available, they're generated, uploaded to an OCI Object Storage bucket, and imported into the management cluster. The OCK image is converted from Qcow2 format to a bootable compute image and saved as a custom compute image.
3. When this process is complete, all Kubernetes Cluster API providers are installed into the management cluster.
4. When the Kubernetes Cluster API providers are started, the Kubernetes Cluster API resources are installed into the management cluster.

5. When the management cluster is ready, the OCI cluster is created and set up, including the compute instances, networking, and a network load balancer.
6. If the OCI cluster is set to be self managed, the Cluster API resources are migrated to the workload cluster.

Creating an OCI Cluster

Create a Kubernetes cluster on OCI using the `oci` provider.

Creating a Kubernetes cluster on OCI using the `oci` provider requires that you first set up the localhost to create a management cluster using the `libvirt` provider, or have a cluster available to use as the management cluster. For information on the management cluster, see OCI Provider.

You must also install and configure the OCI CLI on the localhost to enable access to the OCI compartment where the cluster is created.

You can optionally use a cluster configuration file to specify cluster information such as the number of control plane and worker nodes, the cluster definition file that contains the Cluster API CRs, the Object Storage bucket name (if the default of `ocne-images` isn't used), or any other number of configuration options.

If settings aren't provided in the cluster configuration file options, but are available in the Cluster API provider, create a cluster template file that contains the CRs for the cluster. You can then manually edit these CRs with the Cluster API options and include the template in the cluster configuration file.

> ⚠ **Caution**
>
> The number of control plane nodes must be an odd number (1, 3, 5, and so on) to avoid split brain scenarios with High Availability. The default is 1 control plane node and 0 worker nodes.

1. Set up the localhost to provision clusters using the `oci` provider.

   See Setting Up the OCI Provider.

2. (Optional) Set up a cluster configuration file.

   A cluster configuration contains the cluster specific information to use when creating the cluster. This file overrides any default configuration (set in the `$HOME/.ocne/defaults.yaml` file). A cluster configuration file might include:

   ```
   provider: oci
   name: mycluster
   providers:
     oci:
       compartment: ocid1.compartment.oc1..uniqueID
       vcn: ocid1.vcn.oc1.uniqueID
       loadBalancer:
         subnet1: ocid1.subnet.oc1.uniqueID
         subnet2: ocid1.subnet.oc1.uniqueID
   ```

Or a more complex cluster configuration file might include:

```
provider: oci
proxy:
  httpsProxy: http://myproxy.example.com:2138
  httpProxy: http://myproxy.example.com:2138

noProxy: .example.com,127.0.0.1,localhost,169.254.169.254,10.96.0.0/12,10.2
44.0.0/16
headless: true
name: mycluster
workerNodes: 3
controlPlaneNodes: 3
providers:
  oci:
    profile: MYTENANCY
    selfManaged: false
    imageBucket: my-ocne-images
    compartment: ocid1.compartment.oc1..uniqueID
    vcn: ocid1.vcn.oc1.uniqueID
    loadBalancer:
      subnet1: ocid1.subnet.oc1.uniqueID
      subnet2: ocid1.subnet.oc1.uniqueID
    workerShape:
      shape: VM.Standard.E4.Flex
      ocpus: 2
    controlPlaneShape:
      shape: VM.Standard.E4.Flex
      ocpus: 2
```

For information on cluster configuration files, see Cluster Configuration Files.

3. (Optional) Set up a cluster template file.

   A cluster template contains the Kubernetes Cluster API CRs to create a cluster. You can generate a template using the configuration in a cluster configuration file, and edit the resulting CRs to include options that aren't available as an option in the cluster configuration. For information on creating cluster templates, see Cluster API Templates.

   > **Tip**
   >
   > To configure the cluster to use an existing VCN in a compartment, see Using an Existing VCN.

   Include a link to the cluster template in the cluster configuration file. For example, also include the option:

   ```
   clusterDefinition: /path/template.yaml
   ```

4. Create the cluster.

   Use the `ocne cluster start` command to create a cluster. The syntax is:

   ```
   ocne cluster start
   [{-u|--auto-start-ui} {true|false}]
   ```

```
[{-o|--boot-volume-container-image} URI]
[{-C|--cluster-name} name]
[{-c|--config} path]
[{-n|--control-plane-nodes} integer]
[{-i|--key} path]
[--load-balancer address]
[{-P|--provider} provider]
[{-s|--session} URI]
[{-v|--version} version]
[--virtual-ip IP]
[{-w|--worker-nodes} integer]
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne cluster start --provider oci
```

```
ocne cluster start --config myconfig.yaml
```

Monitoring a Cluster Installation

View the logs for the Kubernetes Cluster API pods to monitor the creation of a Kubernetes cluster on OCI.

You can monitor the deployment of a Kubernetes cluster on OCI by reviewing the logs of several Kubernetes Cluster API pods that are created in the management cluster.

Tip

If you set the cluster to be self managed, the ephemeral cluster is deleted after the deployment succeeds. To view the logs after the ephemeral cluster is deleted, set the cluster's `kubeconfig` file to the workload cluster running on OCI, for example:

```
export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
```

Where *cluster_name* is the name of the workload cluster.

1. Set the location of the `kubeconfig` file for the management cluster.

   In a separate terminal session, set the location of the management cluster's `kubeconfig` file.

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Where *cluster_name* is the name of the management cluster.

2. View the events.

Use the `kubectl get events` command to get information about the events in the namespace in which the cluster is created. The default namespace is `ocne`. For example:

```
kubectl get events --namespace ocne
```

3. View the `capoci-controller-manager` pod logs.

   Use the `kubectl logs` command to view the logs for the pod.

   Copy the command listed and press the `Tab` key to access the full pod name.

   ```
   kubectl logs --namespace cluster-api-provider-oci-system capoci-controller-
   manager
   ```

4. View the `control-plane-capi-controller-manager` pod logs.

   Use the `kubectl logs` command to view the logs for the pod.

   Copy the command listed and press the `Tab` key to access the full pod name.

   ```
   kubectl logs --namespace capi-kubeadm-control-plane-system control-plane-
   capi-controller-manager
   ```

5. View the `bootstrap-capi-controller-manager` pod logs.

   Use the `kubectl logs` command to view the logs for the pod.

   Copy the command listed and press the `Tab` key to access the full pod name.

   ```
   kubectl logs --namespace capi-kubeadm-bootstrap-system bootstrap-capi-
   controller-manager
   ```

OCI Components

Describes the components created in OCI when you create a Kubernetes cluster using the `oci` provider.

You can sign in to the OCI console to view the cluster components that are created. The components created or used are:

- An Object Storage bucket named `ocne-images`, or another existing bucket set in a configuration file.
- OCK images are stored in the Object Storage bucket and named `ocne_boot.qcow2-`*`kube_version`*`-arm64.oci`, by default. They're tagged with the architecture to which they apply.
- The OCK bootable compute images are named `ock`, by default, and copied to the Custom Images area to use when creating instances.
- A Virtual cloud network (VCN) is created and configured for the cluster. The VCN is named using the cluster name.
- A Network load balancer is created using the format *`clustername`*`-apiserver`. A listener is created for the Kubernetes API Server on TCP port 6443. Backend sets are created and include the control plane nodes.
- Compute instances are created using the bootable OCK custom image for the architecture of the instance. The instances use the naming format of *`clustername`*`-md-`*`random_string`*

for worker nodes, and *clustername*-control-plane-*random_string* for control plane nodes.

Connecting to a Cluster

Use the kubectl package to connect to a Kubernetes cluster.

After creating a Kubernetes cluster, a Kubernetes configuration file is created so you can access the cluster using the kubectl command. Install kubectl on the localhost (the host with ocne installed on which you created the cluster). If you install kubectl on a different system, copy the kubeconfig file to the system.

1. Install the kubectl package on the localhost.

   ```
   sudo dnf install kubectl
   ```

2. Set the kubeconfig file location using an environment variable.

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Replace *cluster_name* with the name you used to create the cluster. The default is ocne.

3. (Optional) Persist the environment variable.

   Add the environment variable to the .bashrc file:

   ```
   echo 'export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)'
   >> $HOME/.bashrc
   ```

4. Verify that you can use kubectl to connect to the cluster.

   For example:

   ```
   kubectl get deployments --all-namespaces
   ```

Scale a Cluster

Learn how to scale a Kubernetes cluster on OCI.

You can scale the cluster nodes by using the kubectl scale command, either on the management cluster, or directly in the OCI workload cluster if it's self-managed.

> **Note**
>
> You can also scale cluster nodes by editing the replicas option in the appropriate Kubernetes Cluster API Custom Resource file.

Scaling Worker Nodes in an OCI Cluster

Use the `kubectl scale` command to scale the number of worker nodes in a Kubernetes cluster on OCI.

You can scale the worker nodes in a Kubernetes cluster by running the `kubectl scale` command, either on the management cluster, or on the OCI workload cluster if it's self-managed.

1. Get the name of the `machinedeployments` Custom Resource.

   Use the `kubectl get` command to retrieve this information, for example:

   ```
   kubectl get machinedeployments --namespace ocne
   ```

2. Run the `kubectl scale` command, specifying the new number of worker nodes.

   Set the new number of worker nodes with the `--replicas` option. In this example, the name of the `machinedeployments` Custom Resource is `mycluster-md-0`:

   ```
   kubectl scale machinedeployment mycluster-md-0 --replicas 3 --namespace
   ocne
   ```

3. Wait for the cluster to stabilize before performing any other scaling operations.

   You can monitor the status of the nodes by running the following command on the OCI cluster:

   ```
   watch kubectl get nodes -A
   ```

   The cluster is stable when all the cluster nodes show a status of `Ready`. Exit the command using Ctrl+C.

   The number of worker nodes in the cluster is scaled up or down to the number of replicas you set.

Scaling Control Plane Nodes in an OCI Cluster

Use the `kubectl scale` command to scale the number of control plane nodes in a Kubernetes cluster on OCI.

You can scale the control plane nodes in a Kubernetes cluster by running the `kubectl scale` command, either on the management cluster, or on the OCI workload cluster if it's self-managed.

1. Get the name of the `kubeadmcontrolplane` Custom Resource.

   Use the `kubectl get` command to retrieve this information, for example:

   ```
   kubectl get kubeadmcontrolplane --namespace ocne
   ```

2. Run the `kubectl scale` command, specifying the new number of control plane nodes.

Set the new number of control plane nodes with the `--replicas` option. In this example, the name of the `kubeadmcontrolplane` Custom Resource is `mycluster-control-plane`:

```
kubectl scale kubeadmcontrolplane mycluster-control-plane --replicas 3 --
namespace ocne
```

3. Wait for the cluster to stabilize before performing any other scaling operations.

   You can monitor the status of the nodes by running the following command on the OCI cluster:

   ```
   watch kubectl get nodes -A
   ```

   The cluster is stable when all the cluster nodes show a status of `Ready`. Exit the command using Ctrl+C.

   The number of control plane nodes in the cluster is scaled up or down to the number of replicas you set.

Upgrading an OCI Cluster to a Kubernetes Minor Release

Upgrade an OCI cluster to the next Kubernetes minor release.

For clusters running on OCI, and created using the `oci` provider, upgrade a Kubernetes cluster to the next minor Kubernetes version when an OCK image becomes available for that version. This uses the Kubernetes Cluster API to scale nodes in and out. To perform an in place upgrade, where nodes aren't reprovisioned, use the steps in Upgrading to a Kubernetes Minor Release.

1. Set the location of the management cluster.

   The management cluster contains the Kubernetes Cluster API controllers. The management cluster might be the same as the workload cluster. The upgrade is performed using the management cluster.

   Set the `kubeconfig` file location to the management cluster using an environment variable. For example:

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Replace *cluster_name* with the name of the management cluster.

2. Set the target Kubernetes version.

   Use `ocne cluster stage` command to stage the target Kubernetes version. Use the configuration file used to create the workload cluster with this command. The syntax to use is:

   ```
   ocne cluster stage
   [{-c|--config} path]
   [{-r|--os-registry} registry]
   [{-t|--transport} transport]
   {-v|--version} version
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne cluster stage --version 1.33 --config mycluster.yaml
```

An OCK image that includes the updated Kubernetes minor version is created, and uploaded to the OCI Object Bucket store, then converted to a custom image.

The output of this command prints important information. For example, the output might look similar to:

```
To update KubeadmControlPlane ocne-control-plane in ocne, run:
    kubectl patch -n ocne kubeadmcontrolplane ocne-control-plane --
type=json -p='[{"op":"replace","path":"/spec/version","value":"1.33.5"},
{"op":"replace","path":"/spec/machineTemplate/infrastructureRef/
name","value":"ocne-control-plane-1"},{"op":"add","path":"/spec/
kubeadmConfigSpec/joinConfiguration/patches","value":{"directory":"/etc/
ocne/ock/patches"}}]'

To update MachineDeployment ocne-md-0 in ocne, run:
    kubectl patch -n ocne machinedeployment ocne-md-0 --type=json -
p='[{"op":"replace","path":"/spec/template/spec/version","value":"1.33.5"},
{"op":"replace","path":"/spec/template/spec/infrastructureRef/
name","value":"ocne-md-1"}]'
```

3. Patch the KubeadmControlPlane for control plane nodes.

   Use the `kubectl patch` command to update the KubeadmControlPlane. Use the command printed in the output of the `ocne cluster stage` command. For example:

   ```
    kubectl patch -n ocne kubeadmcontrolplane ocne-control-plane --type=json -
   p='[{"op":"replace","path":"/spec/version","value":"1.33.5"},
   {"op":"replace","path":"/spec/machineTemplate/infrastructureRef/
   name","value":"ocne-control-plane-1"},{"op":"add","path":"/spec/
   kubeadmConfigSpec/joinConfiguration/patches","value":{"directory":"/etc/
   ocne/ock/patches"}}]'
   ```

4. Wait for the control plane nodes to upgrade.

   The control plane nodes are reprovisioned using the new OCK image, with the new version of Kubernetes. This might take some time.

   > **Tip**
   >
   > Monitor new nodes are being provisioned, and old nodes are being removed, using:
   >
   > ```
   > kubectl --namespace namespace get machine
   > ```

   Confirm all nodes are updated in the workload cluster. Set the `kubeconfig` file location to the workload cluster using an environment variable. For example:

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

Replace *cluster_name* with the name of the workload cluster.

List the nodes in the cluster and confirm the `VERSION` column lists the new Kubernetes version number.

```
kubectl get nodes
```

5. Update the MachineDeployment for worker nodes.

   Set the `kubeconfig` file location to the management cluster. For example:

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Replace *cluster_name* with the name of the management cluster.

   Use the `kubectl patch` command to update the MachineDeployment for worker nodes. Use the command printed in the output of the `ocne cluster stage` command. For example:

   ```
   kubectl patch -n ocne machinedeployment ocne-md-0 --type=json -
   p='[{"op":"replace","path":"/spec/template/spec/version","value":"1.33.5"},
   {"op":"replace","path":"/spec/template/spec/infrastructureRef/
   name","value":"ocne-md-1"}]'
   ```

6. Wait for the worker nodes to update.

   The worker nodes are reprovisioned using the new OCK image, with the new version of Kubernetes. This might take some time.

7. Confirm Kubernetes has been upgraded.

   Confirm all nodes are updated in the workload cluster. Set the `kubeconfig` file location to the workload cluster using an environment variable. For example:

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Replace *cluster_name* with the name of the workload cluster.

   List the nodes in the cluster and confirm the `VERSION` column lists the new Kubernetes version number.

   ```
   kubectl get nodes
   ```

Delete a Cluster

Learn how to delete a Kubernetes cluster on OCI.

Deleting a self-managed cluster requires that a second cluster is available to run the Kubernetes Cluster API controllers (a management cluster). This is because the final stages of cluster destruction terminates any remaining compute instances, and, by extension, cluster nodes. When this is complete, no compute instances are available to run the controllers. If a management cluster isn't available, an ephemeral cluster is created to service this need.

Deleting an OCI Cluster

Delete a Kubernetes cluster on OCI.

1. Delete the Kubernetes cluster running on OCI.

   Use the `ocne cluster delete` command to delete the cluster. They syntax to use is:

   ```
   ocne cluster delete
   [{-C|--cluster-name} name]
   [{-c|--config} URI]
   [{-P|--provider} provider]
   [{-s|--session} URI]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster delete --cluster-name mycluster
   ```

   or:

   ```
   ocne cluster delete --config myconfig.yaml
   ```

   The management cluster deletes the OCI workload cluster. If the cluster is self managed, an ephemeral cluster is created and the Kubernetes Cluster API resources are migrated to the ephemeral cluster.

   Tip

   Monitor the cluster deletion process in a separate terminal session. Set the `KUBECONFIG` environment variable to the management cluster.

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Replace *cluster_name* with the name of the management cluster. Then enter:

   ```
   kubectl get cluster -A
   ```

   The cluster object is in the `deleting` state. When the cluster object is no longer available, the delete is completed.

2. Delete the management cluster.

   You might also need to delete the management cluster, depending on the settings in the environment. Use the `ocne cluster delete` command to delete the management cluster. For example:

   ```
   ocne cluster delete --cluster-name mycluster
   ```

3. Delete Oracle Cloud Infrastructure resources.

   You might need to manually remove some Oracle Cloud Infrastructure resources. While the compute instances, Virtual Cloud Networks, and Network load balancers in the cluster are terminated, any OCK images in the Objects Storage bucket, and custom compute images aren't removed. If you don't want to retain those objects, sign in to the Oracle Cloud Infrastructure console and remove them manually.

8
Bring Your Own Provider

Use the `byo` provider to create Kubernetes clusters using bare metal or other virtual instances not provided explicitly by Oracle CNE.

You can make custom installations of the Oracle Container Host for Kubernetes (OCK) image on arbitrary platforms. This means you can create a Kubernetes cluster using bare metal or other virtual instances, not provided explicitly by Oracle CNE. These installations are known as *Bring Your Own* (BYO) installations. You use the `byo` provider to perform these installations.

You can install the OCK image into environments that require manual installation of individual hosts. A common case is a bare metal deployment. Another is a case where a standardized *golden image* for an OS is required. This install type is intended to cover all cases where deploying the standard OS boot image isn't possible.

This installation process is used to create new Kubernetes clusters or expand existing ones. This installation type leverages the Anaconda and Kickstart installation options of Oracle Linux to deploy OSTree content onto a host.

The BYO installation consists of a handful of components, spread across several Oracle CNE CLI commands.

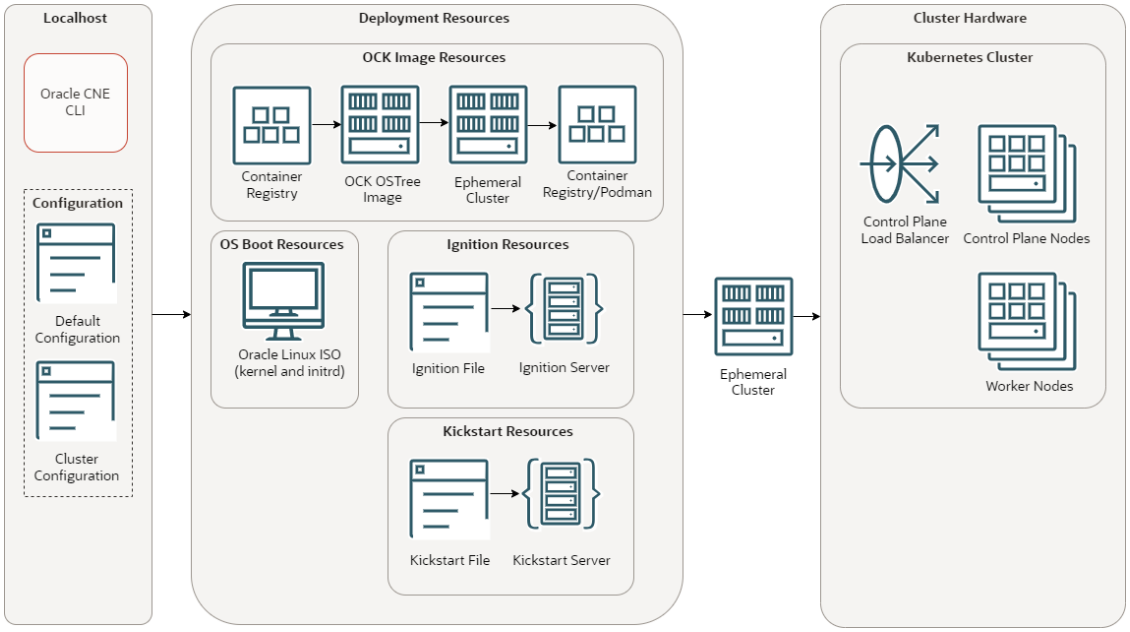
- The `ocne image create` command is used to download OSTree content from official Oracle CNE sources, and convert them into a format that can be used for a custom installation. It also creates an OSTree archive server.
- The `ocne image upload` command is used to copy the OSTree archive server to a container registry. You can also use Podman to serve the OSTree archive locally if you don't want to use a container registry. You can load the image into any target available with the Open Container Initiative transports and formats. See `containers-transports(5)` for available options.
- The `ocne cluster start` command generates Ignition content that's consumed by the newly installed host during boot. This Ignition information is used to start a new Kubernetes cluster. You specify what you want to include in the Ignition configuration.
- The `ocne cluster join` command generates Ignition content that's used to add nodes to an existing Kubernetes cluster.

For more information on OSTree, see the upstream OSTree documentation.

For more information on Ignition, see the upstream Ignition documentation.

BYO installations of the OCK image use an OSTree archive with Anaconda and Kickstart to create bootable media. When the base OS installation is complete, Ignition is used to complete the first-boot configuration and provision Kubernetes services on the host.

Figure 8-1 BYO Cluster



The BYO cluster architecture has the following components:

- **CLI**: The CLI used to create and manage Kubernetes clusters. The `ocne` command.
- **Default configuration**: A YAML file that contains configuration for all `ocne` commands.
- **Cluster configuration**: A YAML file that contains configuration for a specific Kubernetes cluster.
- **Container registry**: A container registry used to pull the OCK OSTree images. The default is the Oracle Container Registry.
- **OCK OSTree Image**: The OCK OSTree image pulled from the container registry.
- **Ephemeral cluster**: A temporary Kubernetes cluster used to perform a CLI command.
- **Container registry/Podman**: A container registry or container server, such as Podman, used to serve the OCK OSTree images.
- **Oracle Linux ISO**: An ISO file to serve the kernel and initrd to use for the OS on nodes.
- **Ignition file**: An Ignition file, generated by the CLI, used to join nodes to a cluster.
- **Ignition server**: The Ignition file, loaded into a method that serves Ignition files.
- **Kickstart file**: A Kickstart file that provides the location of the OCK OSTree image, Ignition file, and the OS kernel and initrd.
- **Kickstart server**: The Kickstart file, loaded into a method that servers Kickstart files.
- **Ephemeral cluster**: A temporary Kubernetes cluster used to perform a CLI command.
- **Control plane load balancer**: A load balancer used for High Availability (HA) of the control plane nodes. This might be the default internal load balancer, or an external one.
- **Control plane nodes**: Control plane nodes in a Kubernetes cluster.
- **Worker nodes**: Worker nodes in a Kubernetes cluster.

OS Image

Describes the OS images used to create Kubernetes nodes using the `byo` provider.

You need the following to perform a BYO installation:

- An Oracle Linux kernel to boot.
- An initrd (initial ramdisk) that matches the boot kernel.
- A root file system that can run Anaconda and Kickstart.
- A method to perform an automated installation using Kickstart.
- A method to serve the OCK OSTree archive.
- A method to serve Ignition files.

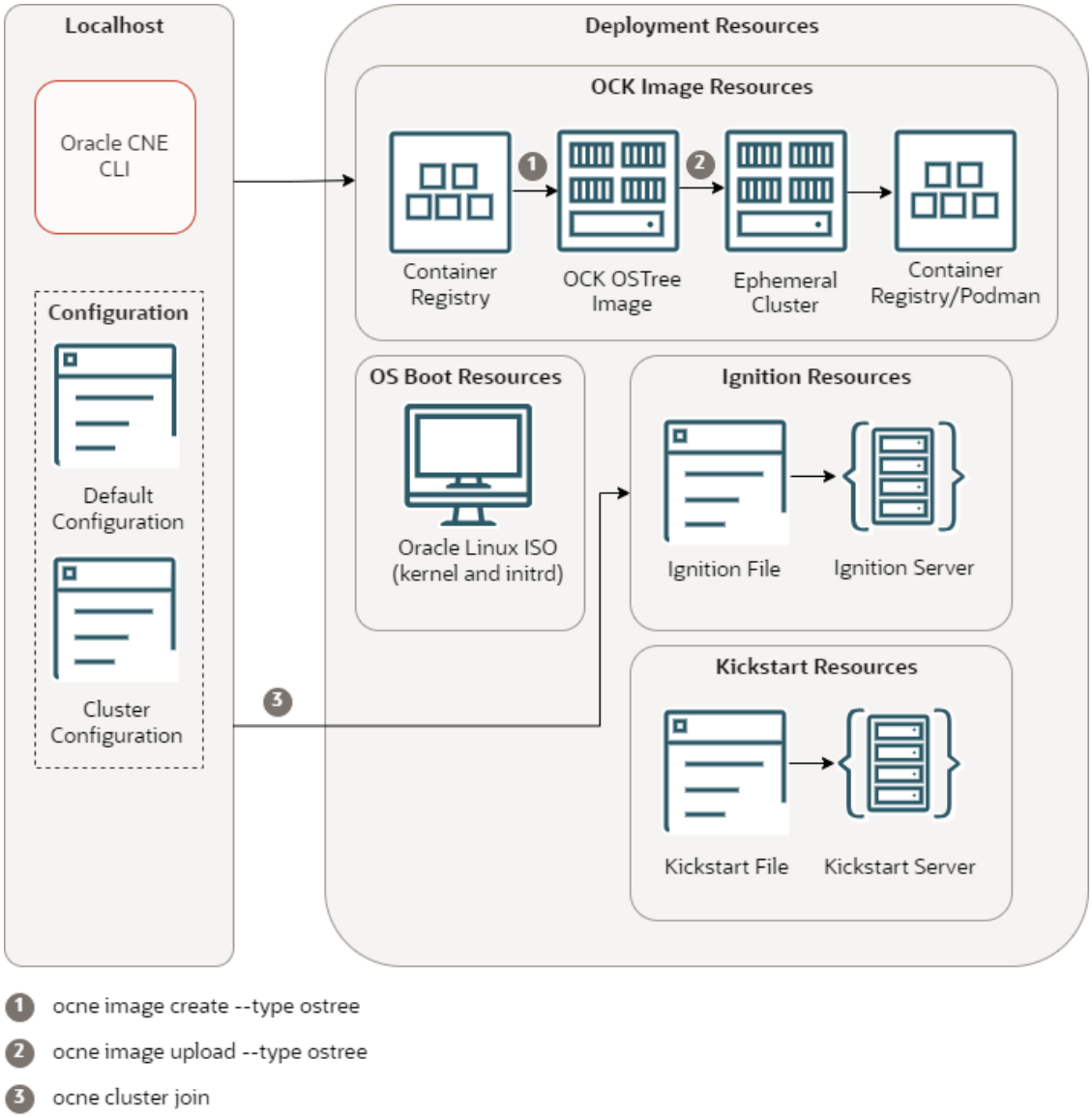
An easy way to achieve the first four points is to download an Oracle Linux ISO file. For information about the available ISO files, see Oracle Linux ISO Images.

A Kickstart file defines an automated installation using a local OSTree archive server. For information on creating a Kickstart file, see Oracle Linux 9: Installing Oracle Linux or Oracle Linux 8: Installing Oracle Linux.

The CLI can generate a container image that serves an OSTree archive over HTTP. The container image can be served using a container runtime such as Podman, or inside a Kubernetes cluster. You can also generate an OSTree archive manually and serve it over any HTTP server.

Ignition files can be served using any of the platforms listed in the upstream Ignition documentation. You could also embed the Ignition configuration file directly on to the root file system of the host if the installation is done reasonably close to when the Ignition configuration was generated.

Figure 8-2 OCK Images in a Bring Your Own Cluster



- **CLI**: The CLI used to create and manage Kubernetes clusters. The `ocne` command.
- **Default configuration**: A YAML file that contains configuration for all `ocne` commands.
- **Cluster configuration**: A YAML file that contains configuration for a specific Kubernetes cluster.
- **Container registry**: A container registry used to pull the OCK OSTree images. The default is the Oracle Container Registry.
- **OCK OSTree Image**: The OCK OSTree image pulled from the container registry.
- **Ephemeral cluster**: A temporary Kubernetes cluster used to perform a CLI command.
- **Container registry/Podman**: A container registry or container server, such as Podman, used to serve the OCK OSTree images.
- **Oracle Linux ISO**: An ISO file to serve the kernel and initrd to use for the OS on nodes.

- **Ignition file**: An Ignition file, generated by the CLI, used to join nodes to a cluster.
- **Ignition server**: The Ignition file, loaded into a method that serves Ignition files.
- **Kickstart file**: A Kickstart file that provides the location of the OCK OSTree image, Ignition file, and the OS kernel and initrd.
- **Kickstart server**: The Kickstart file, loaded into a method that servers Kickstart files.

Oracle Linux ISO Images

Describes the Oracle Linux ISO image used for OS images with the `byo` provider.

You can use any of the available Oracle Linux ISO images as the basis for the OS image. Only the kernel, initrd, and root file system from the media is used. All installation content comes from the OCK OSTree archive. The ISO images are available on the Oracle Linux yum server.

We recommend you use a UEK boot ISO file as these are smaller and include all the OS components required.

OSTree Archive Server

Describes the OSTree archive used for building an OS image with the `byo` provider.

A container image that contains the OSTree archive can be generated using the `ocne image create` command. For example:

```
ocne image create --type ostree
```

A Kubernetes cluster is used to download and generate the OSTree archive image. Any running cluster can be used. To specify which cluster to use, set the `KUBECONFIG` environment variable, or use the `--kubeconfig` option of `ocne` commands. If no cluster is available, a cluster is created automatically using the `libvirt` provider, with the default configuration.

The image might take some time to download and generate. When completed, the image is available in the `$HOME/.ocne/images/` directory.

Use the `ocne image upload` command to upload the image to a location where it can be used. Container images can be uploaded using any transport provided by `libcontainer`. See `containers-transports(5)` for the list of transports. A transport is always required.

Typically, the image is uploaded to a container registry. For example:

```
ocne image upload --type ostree --file $HOME/.ocne/images/ock-1.33-amd64-
ostree.tar --destination docker://myregistry.example.com/ock-ostree:latest --
arch amd64
```

To load the container image into the local image cache, use the Open Container Initiative image loading facility built in to a container runtime or other some other tool that performs the same task. For example, to load the image into Podman:

```
podman load < $HOME/.ocne/images/ock-1.33-arm64-ostree.tar
```

Creating an OSTree Image for the Bring Your Own Provider

Create an OSTree image for the Bring Your Own (`byo`) provider. Then upload the image to a container registry so it can be used as the boot disk for Virtual Machines (VMs).

You can load the image into several destinations. This example show you how to load the image into a container registry. For information on setting up a container registry, see Oracle Linux: Podman User's Guide.

You can load the image into any target available with the Open Container Initiative transports and formats. See `containers-transports(5)` for available options.

1. (Optional) Set the location of the `kubeconfig` file for an existing cluster.

   A Kubernetes cluster is required to perform some steps. You can use an existing cluster for this purpose by setting the location of the `kubeconfig` file.

   You can set this using the `KUBECONFIG` environment variable, or using the `--kubeconfig` option with `ocne` commands. You could also set this in a configuration file.

   If you don't set the location of the `kubeconfig` file, an ephemeral cluster is created using the `libvirt` provider when required.

2. (Optional) Set up the `libvirt` provider.

   If you don't set the location of an existing Kubernetes cluster, set up the localhost to provision ephemeral clusters using the `libvirt` provider. For information on setting up the `libvirt` provider, see Setting Up the libvirt Provider.

3. Create an OSTree image.

   Use the `ocne image create` command to create a OSTree image. The syntax is:

   ```
   ocne image create
   {-a|--arch} arch
   [{-t|--type} provider]
   [{-v|--version} version]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne image create --type ostree --arch arm64
   ```

   This command might take some time to complete.

   The Kubernetes cluster generates the OSTree image, and the image is saved to the `$HOME/.ocne/images/` directory.

4. Upload the OSTree image to a container registry.

   Use the `ocne image upload` command to upload the image to a container registry. The syntax is:

   ```
   ocne image upload
   {-a|--arch} arch
   [{-b|--bucket} name]
   [{-c|--compartment} name]
   [--config path]
   ```

```
[{-d|--destination} path]
{-f|--file} path
{-i|--image-name} name
{-t|--type} provider
{-v|--version} version
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne image upload --type ostree --file $HOME/.ocne/images/ock-1.33-amd64-
ostree.tar --destination docker://myregistry.example.com/ock-ostree:latest
--arch amd64
```

The Kubernetes cluster uploads the OSTree image. A sign in prompt is provided if credentials aren't set for the target container registry. The image is uploaded to the container registry. The image can now be used to create VMs to use in a Kubernetes cluster.

Creating a Bring Your Own Cluster

Create a Kubernetes cluster using the `byo` provider.

These steps provide a high level overview of the process to create a Kubernetes cluster using the `byo` provider. Many options are available to perform each step, so we don't provide detailed steps. You can decide which methods and options to use in many of these steps.

> **⚠ Caution**
>
> The steps and commands shown here are provided as examples only and must be adapted to suit a specific deployment.

1. (Optional) Set the location of the `kubeconfig` file for an existing cluster.

   A Kubernetes cluster is required to perform some steps. You can use an existing cluster for this purpose by setting the location of the `kubeconfig` file.

   You can set this using the `KUBECONFIG` environment variable, or using the `--kubeconfig` option with `ocne` commands. You could also set this in a configuration file.

   If you don't set the location of the `kubeconfig` file, an ephemeral cluster is created using the `libvirt` provider when required.

2. (Optional) Set up the `libvirt` provider.

   If you don't set the location of an existing Kubernetes cluster, set up the localhost to provision ephemeral clusters using the `libvirt` provider. For information on setting up the `libvirt` provider, see Setting Up the libvirt Provider.

3. Prepare the automated Oracle Linux installation.

   Decide on the method to perform an automated install of Oracle Linux on the hosts using a Kickstart file. For example, you might want to use a network drive, a web server, or a USB drive. You only need to provision the kernel and initrd (initial ramdisk) that matches the boot kernel.

We recommend you use an Oracle Linux UEK boot ISO file as the boot media as it contains the required kernel and initrd, in a smaller file size. Download Oracle Linux ISO files from the Oracle Linux yum server.

Prepare the Oracle Linux boot media using the method you select.

For more information about the automated installation options for Oracle Linux, see Oracle Linux 9: Installing Oracle Linux or Oracle Linux 8: Installing Oracle Linux.

4. Create an OSTree archive container image.

   Use the `ocne image create` command to generate an OSTree archive container image, then upload it to somewhere it can be used during the installation using the `ocne image upload` command. For information on creating an OSTree image and uploading it to a container registry, see Creating an OSTree Image for the Bring Your Own Provider.

   If you don't have a container registry, you might prefer to load the OSTree archive image to a local container runtime. For example, to load an OSTree archive file for an arm64 image to Podman on the localhost, you might use:

   ```
   podman load < $HOME/.ocne/images/ock-1.33-arm64-ostree.tar
   ```

5. Create a container to serve the OSTree image.

   Use any container runtime you like, including on a Kubernetes cluster. For example, to use an image loaded into Podman on the localhost, you might use:

   ```
   podman run -d --name ock-content-server -p 8080:80 localhost/ock-
   ostree:latest
   ```

6. Set up the location of Ignition files.

   Decide how you want to make the Kubernetes cluster Ignition files available.

   An Ignition file must be available to all hosts during their first boot. Ignition files can be served using any of the platforms listed in the upstream Ignition documentation, for example, using a Network File Server (NFS), or a web server. You could also embed the Ignition configuration file directly on to the root file system of the host if the installation is done reasonably close to when the Ignition configuration was generated.

7. Create a Kickstart file.

   A Kickstart file defines an automated installation of Oracle Linux. Include the information to use the OSTree in the installation in the Kickstart file. For information on creating a Kickstart file, see Oracle Linux 9: Installing Oracle Linux or Oracle Linux 8: Installing Oracle Linux.

   The Kickstart file must be made available during the installation. It might be useful to include the Kickstart file in the same location as the Ignition file, for example, using NFS, or a web server.

   The Kickstart file must include the OSTree image information, and the information about the location of the Ignition file, if you aren't embedding this information on the root file system. The Ignition file is created later. For example, a bare metal installation might use something similar to:

   ```
   ...
   services --enabled=ostree-remount
   bootloader --append "rw ip=dhcp rd.neednet=1 ignition.platform.id=metal
   ignition.config.url=http://myhost.example.com/ignition.ign
   ignition.firstboot=1"
   ```

```
ostreesetup --nogpg --osname ock --url http://myregistry.example.com/
ostree --ref ock
%post
%end
```

For information about OSTree, see the upstream OSTree documentation.

8. Create a Kubernetes cluster configuration file.

   Generate a cluster configuration file that defines the Kubernetes cluster to create. Ensure the provider is set to `byo`. In this example, a virtual IP of `192.168.124.230` is used. The virtual IP must be an unused IP address in the network to be used for the Kubernetes Cluster API Server.

   For example:

   ```
   provider: byo
   name: byocluster
   virtualIp: 192.168.122.230
   providers:
     byo:
       networkInterface: enp1s0
   ```

   For information on what can be included in the cluster configuration file, see Cluster Configuration Files.

9. Generate and expose the Kubernetes cluster Ignition information.

   The `byo` provider doesn't provision any infrastructure resources. Unlike other providers that create Kubernetes cluster nodes automatically, the `byo` provider generates the Ignition configuration that applies to the node type and cluster configuration.

   Use the `ocne cluster start` command to generate the Ignition information that starts the first control plane node. The syntax is:

   ```
   ocne cluster start
   [{-u|--auto-start-ui} {true|false}]
   [{-o|--boot-volume-container-image} URI]
   [{-C|--cluster-name} name]
   [{-c|--config} path]
   [{-n|--control-plane-nodes} integer]
   [{-i|--key} path]
   [--load-balancer address]
   [{-P|--provider} provider]
   [{-s|--session} URI]
   [{-v|--version} version]
   [--virtual-ip IP]
   [{-w|--worker-nodes} integer]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   Use the cluster configuration file with this command, and save the output to an Ignition file. For example:

   ```
   ocne cluster start --config myconfig.yaml > ignition.ign
   ```

> **Tip**
>
> The Ignition file can be inspected using the `jq` utility, for example:
>
> ```
> jq . < ignition_file.ign
> {
>   "ignition": {
>     "config": {
>       "replace": {
>         "verification": {}
>       }
>     },
>     "proxy": {},
>     "security": {
>       "tls": {}
> ...
> ```

Expose the Ignition file so it's available during the installation.

10. Boot the first control plane node.

    Install an Oracle Linux host using the Kickstart file. The Kickstart file uses the Oracle Linux boot ISO, the OSTree image, and the Kubernetes cluster Ignition file to set up the host. This host is to be used as the first control plane node to start the Kubernetes cluster.

11. Start the Kubernetes cluster with the control plane node.

    Use the `ocne cluster start` command to start the Kubernetes cluster and install any configured software into the cluster. The syntax is:

    ```
    ocne cluster start
    [{-u|--auto-start-ui} {true|false}]
    [{-o|--boot-volume-container-image} URI]
    [{-C|--cluster-name} name]
    [{-c|--config} path]
    [{-n|--control-plane-nodes} integer]
    [{-i|--key} path]
    [--load-balancer address]
    [{-P|--provider} provider]
    [{-s|--session} URI]
    [{-v|--version} version]
    [--virtual-ip IP]
    [{-w|--worker-nodes} integer]
    ```

    For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

    Use the cluster configuration file to start the cluster. For example:

    ```
    ocne cluster start --config myconfig.yaml
    ```

    The control plane node is used to start the cluster, install the UI, application catalog, and any applications. The control plane node is now a single node Kubernetes cluster, configured as specified in the cluster configuration file.

12. Install `kubectl`.

    For information on installing `kubectl` and setting up the `kubeconfig` file, see Connecting to a Cluster. Set the `kubeconfig` file location to the new cluster. The location of this file is displayed in output of the previous step, when the cluster was created.

13. Confirm the control plane node is added to the cluster.

    Use the `kubectl get nodes` command to confirm the control plane node is added to the cluster. This might take a few moments.

    ```
    kubectl get nodes
    ```

14. Generate and expose an Ignition file for a worker node.

    Use the `ocne cluster join` command to generate the Ignition information that joins a worker node to the cluster. The syntax is:

    ```
    ocne cluster join
    [{-c|--config} path]
    [{-d|--destination} path]
    [{-N|--node} name]
    [{-P|--provider} provider]
    [{-r|--role-control-plane}]
    ```

    For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

    Use the cluster configuration file and save the output to a file. For example:

    ```
    ocne cluster join --kubeconfig $HOME/.kube/kubeconfig.byocluster --config
    myconfig.yaml > mycluster-join-w.ign
    ```

    Important

    Set the location of the BYO cluster using the `--kubeconfig` command option. This option is required for this command.

    A token to join the cluster is generated by this command and the command to use this token is displayed. The token is included in the Ignition file. You use this token to join the worker node to the cluster in the next step.

    Expose the Ignition file using the same method you used for the first control plane node. You can either overwrite the Ignition file for the first control plane node, or edit the Kickstart file to set the location of the worker node Ignition file.

15. Create a Kubernetes bootstrap token for a worker node.

    Use the token printed from the `ocne cluster join` command in the previous step to join the worker node to the cluster.

    ```
    ocne cluster console --node node_name --direct -- kubeadm token create
    token
    ```

This command connects to the control plane node's console using the `ocne cluster console` command, and creates the token in the single node cluster that's running on that node. The token is displayed in the output.

> **Tip**
>
> You can reuse this bootstrap token to add more nodes within the token expiration time allocated by Kubernetes. Or you can create a token for each node.

16. Boot the worker node.

    Install an Oracle Linux host using the Kickstart file. This host is to be used as the first worker node in the Kubernetes cluster.

17. Confirm the worker node is added to the cluster.

    Use the `kubectl get nodes` command to confirm the worker node is added to the cluster. This might take a few moments.

    ```
    kubectl get nodes
    ```

18. Generate and expose an Ignition file for a second control plane node.

    Use the `ocne cluster join` command to generate the Ignition information that joins a control plane node to the cluster. For example:

    ```
    ocne cluster join --kubeconfig $HOME/.kube/kubeconfig.byocluster --role-
    control-plane --config myconfig.yaml > mycluster-join-cp.ign
    ```

    An encrypted certificate bundle and token to join the cluster are generated and displayed by this command. The token is included in the Ignition file. You use this token and certificate bundle to join the control plane node to the cluster in the next step.

    Expose the Ignition file.

19. Create a Kubernetes bootstrap token for the second control plane node.

    When adding a control plane node, two things need to be created: a join token, and an encrypted certificate bundle. These were dynamically created by the `ocne cluster join` command in the previous step.

    Create the certificate bundle:

    ```
    ocne cluster console --node node_name --direct -- kubeadm init phase
    upload-certs --certificate-key certificate-key --upload-certs
    ```

    This command connects to the control plane node's console using the `ocne cluster console` command, and creates the certificate bundle in the cluster.

    Create the join token:

    ```
    ocne cluster console --node node_name --direct -- kubeadm token create
    token
    ```

20. Boot the second control plane node.

    Install an Oracle Linux host using the Kickstart file. This host is to be used as the second control plane node in the Kubernetes cluster.

21. Confirm the control plane node is added to the cluster.

    ```
    kubectl get nodes
    ```

    While control plane nodes are joining the cluster, there might be periodic errors reported by `kubectl` as control plane components adapt to the new node. These errors stop after a few seconds if the node is correctly added to the cluster.

22. Repeat the process to add worker or control plane nodes as needed.

Connecting to a Cluster

Use the `kubectl` package to connect to a Kubernetes cluster.

After creating a Kubernetes cluster, a Kubernetes configuration file is created so you can access the cluster using the `kubectl` command. Install `kubectl` on the localhost (the host with `ocne` installed on which you created the cluster). If you install `kubectl` on a different system, copy the `kubeconfig` file to the system.

1. Install the `kubectl` package on the localhost.

   ```
   sudo dnf install kubectl
   ```

2. Set the `kubeconfig` file location using an environment variable.

   ```
   export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)
   ```

   Replace *cluster_name* with the name you used to create the cluster. The default is `ocne`.

3. (Optional) Persist the environment variable.

   Add the environment variable to the `.bashrc` file:

   ```
   echo 'export KUBECONFIG=$(ocne cluster show --cluster-name cluster-name)'
   >> $HOME/.bashrc
   ```

4. Verify that you can use `kubectl` to connect to the cluster.

   For example:

   ```
   kubectl get deployments --all-namespaces
   ```

Migrate Cluster Nodes

Describes migrating a node between Kubernetes cluster using the `byo` provider.

Nodes in a Kubernetes cluster can be migrated from one cluster to another.

Migrating a node from one cluster to another is most useful when it's not feasible to coordinate adding the key material necessary to join a node to an existing cluster at the same time the host is provisioned. In these cases, the easiest path forward is to create a single node cluster and move the node to the target cluster. In this way, it's possible to stitch together several small clusters into a single, larger, cluster.

> **Note**
>
> A host running the OCK image always includes a Kubernetes cluster, even if it's a single node cluster.

Migrating nodes between clusters serves two use cases. Nodes can be reallocated to adjust cluster capacity based on short term requirements, without the need to fully reprovision the node. Or, nodes can be provisioned, and added to a cluster at a later time.

Migrating nodes between clusters might be useful where infrastructure management schemes don't have any service level agreements between when a request to provision a system, or set of systems, is created, and when that request is fulfilled. For example, a cluster administrator requires 8 nodes and the IT department fulfills the request for the resources on an unknown timetable, handing over the access information after the nodes are available. The administrator isn't necessarily told when the resources are provisioned, or even provided with a timetable. This means it's difficult to guarantee that any key material required for a node to join a cluster (certificate keys, and join tokens) is valid when the new systems first boot. Even if coordination were possible, the IT department must also understand Kubernetes cluster management enough to perform any manual cluster provisioning. This isn't a feasible work flow for the typical Bring Your Own (BYO) cluster.

To solve the use case described, the intended path is to ask the IT department to make some number of nodes. Each of those systems boot automatically to single node clusters. The administrator then gathers these nodes and builds the cluster topology they require by migrating each node into the larger cluster.

Two Kubernetes clusters are required. While they can be installed using any method, the intended use case is to either boot a host with OCK installed, but unconfigured, or to merge two BYO clusters.

You can migrate nodes with similar configurations, and Kubernetes versions, between clusters. The Kubernetes version on the source and target clusters doesn't need to be the same, but it must be *close enough*. Close enough means that they must be in the same minor Kubernetes release, or the previous minor release. For example, they must both be running Kubernetes Release 1.33.*x*, or running one minor release earlier, such as Release 1.32.*x*.

The `ocne cluster join` command is used to migrate a node from one cluster to another. You provide the source and target cluster configuration information (the `kubeconfig` file), and the name of the node to migrate. The name of the node must be the same as displayed when using the `kubectl get nodes` command. The name of the node isn't changed when it's migrated. The node name stays the same.

Nodes are migrated as worker nodes, unless you specify the `--role-control-plane` node option of the `ocne cluster join` command.

You can also use the `ocne cluster join` command to generate the Ignition information to join a node to a BYO cluster. For example:

```
ocne cluster join --kubeconfig $HOME/.kube/kubeconfig.mycluster --config
byo.yaml > worker.ign
```

> **⚠ Caution**
>
> Migrating the last control plane node in a cluster destroys that cluster. Ensure you migrate, or remove, all worker nodes first.

Migrating a Cluster Node

Migrate a Kubernetes cluster node, created with the `byo` provider, to another cluster.

Migrating a node from one cluster to another requires the `kubeconfig` file for each cluster, and the name of the node to migrate. The name of the node must be the same as displayed using the `kubectl get nodes` command.

Set the location of the source cluster using the `--kubeconfig` command option. This option is required for this command.

1. Get the `kubeconfig` files.

   You most likely have the `kubeconfig` file for the target cluster, but in some situations you might not have this information for the source cluster node (the node to migrate). You can get the Kubernetes configuration information by logging into the node as the `ocne` user using SSH. When you log in, the Kubernetes configuration information for the node is displayed. Save this information to a local file to use for the source `kubeconfig` file.

   For information the `ocne` user credentials, see OCK Image User.

2. Join the node to the target cluster.

   Use the `ocne cluster join` command to migrate the node to the target cluster. The syntax to use is:

   ```
   ocne cluster join
   [{-c|--config} path]
   [{-d|--destination} path]
   [{-N|--node} name]
   [{-P|--provider} provider]
   [{-r|--role-control-plane}]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   To migrate a node from one cluster to another BYO cluster:

   ```
   ocne cluster join --kubeconfig $HOME/.kube/kubeconfig.mycluster --provider
   byo --node source-worker-1  --destination $HOME/.kube/
   kubeconfig.targetcluster
   ```

   To migrate a node from one cluster to another BYO cluster, and assign it as a control plane node:

   ```
   ocne cluster join --kubeconfig $HOME/.kube/kubeconfig.mycluster --provider
   byo --node source-worker-1 --destination $HOME/.kube/
   kubeconfig.targetcluster --role-control-plane
   ```

Deleting a Cluster

Delete a Kubernetes cluster.

Use the `ocne cluster delete` command to delete the cluster. They syntax to use is:

```
ocne cluster delete
[{-C|--cluster-name} name]
[{-c|--config} URI]
[{-P|--provider} provider]
[{-s|--session} URI]
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

Example 8-1 Delete a cluster using the cluster name

To delete a cluster named `mycluster`:

```
ocne cluster delete --cluster-name mycluster
```

Example 8-2 Delete a cluster using a configuration file

To delete a cluster created using a configuration file:

```
ocne cluster delete --config myconfig.yaml
```

9
UI

Introduces the Oracle CNE User Interface (UI).

The Oracle CNE UI provides a web-based interface to manage the maintenance and installation of Kubernetes cluster resources, and applications.

The UI runs in the Kubernetes cluster as a deployment named `ui`, running in the `ocne-system` namespace. A deployment named `ocne-catalog` also runs in the `ocne-system` namespace to serve the application catalog.

The UI is based on the open source Kubernetes UI Headlamp application. For more information on the Headlamp project, see the upstream Headlamp documentation.

Creating an Access Token

Create an access token to authenticate a connection to the UI.

The UI is deployed into the cluster as a Kubernetes deployment named `ui`, running in the `ocne-system` namespace. A Kubernetes service to access this deployment is also created. The service is also named `ui` and running in the `ocne-system` namespace. To connect to the service, generate an access token.

1. Create an access token.

   Create an access token for the `ui` service:

   ```
   kubectl --namespace ocne-system create token ui
   ```

   The token is displayed.

2. Save the token.

   Save the token in a secure location so you can use it to authenticate a connection when you access the UI.

Exposing the UI Using Port Forwarding

Expose the UI service using port forwarding.

Port-forwarding is a convenient way of exposing the UI service on the localhost for debugging and troubleshooting in a development environment.

> ⚠ **Caution**
>
> We don't recommend you use port-forwarding to expose the UI in a production environment.

1. Generate UI access token.

Generate the access token needed for authenticating a connection to the UI. For information on creating an access token, see Creating an Access Token.

2. Set up port forwarding.

   Set up port forwarding by running the following command:

   ```
   kubectl port-forward --namespace ocne-system service/ui 8443:443
   ```

   > **Note**
   >
   > You must let the `kubectl port-forward` command continue to run for the time you need access to the UI.

3. Access to the UI.

   You can access the UI on the localhost using a web browser. Open a browser session and enter the following address:

   ```
   https://127.0.0.1:8443
   ```

   The **Authentication** page is displayed.

   > **Tip**
   >
   > You can access the UI on the localhost from a remote machine by using local port forwarding. For example, to enable incoming connections to localhost on port `9898`, enter the following command on the remote machine:
   >
   > ```
   > ssh -L 9898:127.0.0.1:8443 myuser@myhost.example.com
   > ```

4. Enter the access token.

   Enter the access token into the **ID token** field on the Security page and click **AUTHENTICATE**.

   The UI application uses the token to authenticate the connection and the **Overview** page appears.

Adding the UI and Application Catalogs into a Cluster

Install the UI and application catalogs into an existing Oracle CNE Release 1 Kubernetes cluster.

Use these steps to install the UI and default application catalog into an Oracle CNE Release 1 Kubernetes cluster. Any extra catalogs set up in a configuration file are also installed.

The target cluster must be healthy. You must provide the `kubeconfig` file for the target cluster.

1. Install the UI and catalogs.

Use the `ocne cluster start` with the `--provider` option set to `none` and specify the location of the target cluster's `kubeconfig` file. The syntax is:

```
ocne cluster start
[{-u|--auto-start-ui} {true|false}]
[{-o|--boot-volume-container-image} URI]
[{-C|--cluster-name} name]
[{-c|--config} path]
[{-n|--control-plane-nodes} integer]
[{-i|--key} path]
[--load-balancer address]
[{-P|--provider} provider]
[{-s|--session} URI]
[{-v|--version} version]
[--virtual-ip IP]
[{-w|--worker-nodes} integer]
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne cluster start --provider none --kubeconfig $HOME/.kube/
kubeconfig.ocne19
```

2. Set the `kubeconfig` file location using an environment variable.

   To access the cluster, set an environment variable to access the Oracle CNE Release 1 cluster. For example:

   ```
   export KUBECONFIG=$HOME/.kube/kubeconfig.ocne19
   ```

3. Configure access to the UI.

   Follow the prompts in the output of the `ocne cluster start` command to set up an access token for the UI. For more information on setting up an access token, see Creating an Access Token.

4. Verify the catalogs are installed.

   Use the `ocne catalog list` command to verify the Oracle application catalog, and any external catalogs are installed. The syntax is:

   ```
   ocne catalog {list|ls}
   ```

   For example:

   ```
   ocne catalog list
   ```

5. Verify the list of available applications.

   Use the `ocne catalog search` command to see the list of applications available in each catalog. The syntax is:

   ```
   ocne catalog search
   [{-N|--name} name]
   [{-p|--pattern} pattern]
   ```

For example, to list all the applications available in the Oracle catalog:

```
ocne catalog search
```

10

Cluster Administration

Describes administration of Kubernetes clusters using the CLI.

This chapter contains information on using the CLI to administer Kubernetes clusters.

Cluster Updates
The `ocne node update` command is used to update the Oracle Container Host for Kubernetes (OCK) image on nodes in the cluster. Updating the OCK image is used for patch updates, and for minor Kubernetes updates.

Cluster Backups
Backups of a cluster can be done with the `ocne cluster backup` command.

Cluster Analysis
The `ocne cluster dump` and `ocne cluster analyze` commands are used to create and analyze a dump of cluster and node data from a Kubernetes cluster. Analyzing a cluster is useful for debugging and getting detailed information about a cluster.

OS Console
The `ocne cluster console` command is used to connect to the OS console of a node in a cluster. The console provides a method to connect to the host in a `chroot`ed environment to perform debugging or inspection of the host's OS.

Cluster Updates

Learn how to update a Kubernetes cluster by updating the Oracle Container Host for Kubernetes (OCK) image on each Kubernetes node.

This section shows you how to update nodes to the latest Kubernetes patch release, or to update them to the next Kubernetes minor release.

Patch releases include errata updates and might include Common Vulnerabilities and Exposures (CVE) fixes, Kubernetes updates, OS updates, and so on. An update to the next Kubernetes minor version is performed in the same way as patch updates, with one extra step to set the Kubernetes version number.

Oracle CNE delivers all updates through updated Oracle Container Host for Kubernetes (OCK) images. Updates are delivered through an OCK image that's specific to the Kubernetes minor version, for example for Kubernetes Release 1.33.

Each node periodically polls the container registry to check for updates to the OCK image it's running, or for an image for the target Kubernetes version if you're upgrading Kubernetes. When you set the Kubernetes version for an upgrade, the image for that version is pulled and staged on the nodes in the cluster. Patch updates are downloaded to each node automatically and don't need to be staged before a node update.

When an update is available, use the `ocne node update` command to reboot a node to use the new image. Running the `ocne node update` command for a node completes the following actions:

1. The node is drained (using the `kubectl drain` command) from the cluster. This evicts the pods from the node.
2. The host OCK image is installed on the node, and the node is restarted.
3. The node is returned to the cluster (using the `kubectl uncordon` command) and is made available to run pods.

Update nodes sequentially, starting with the control plane nodes.

> **Tip**
>
> To save time, you can start the update process as soon as one of the control plane nodes has been annotated as having an update available.

You can update a Highly Available cluster without bringing the cluster down. As one control plane node is taken offline, another control plane node takes control of the cluster. In a cluster with a single control plane node, the control plane node is offline for a short time while the update is performed.

If applications are running on more than one worker node, they remain up, and available, during an update.

Best Practices for Cluster Updates

Learn about best practices for updating Kubernetes clusters.

The following list describes best practices to be followed when updating a Kubernetes cluster in a production environment:

Back up etcd database
In the rare event of an OCK image update failure, the update is rolled back to the previous OCK image. The host reboots into the previous OCK image and rejoins the cluster. However, despite such safeguards being in place, we recommend you follow best practice and back up the `etcd` database before updating a cluster.

Update control plane nodes before worker nodes
Always update the nodes in the control plane first, one node at a time. Confirm the update on the control plane node you're working on has completed, and that the node has rejoined the cluster, before starting an update on another node.

Update nodes immediately after staging a new Kubernetes version
When you stage an image for a Kubernetes minor upgrade, the cluster stops polling for patch updates for the current version. Therefore, apply the upgrade as soon as possible.

Check Kubernetes rules
Certain Kubernetes configurations might prevent a node from being taken offline for upgrade. For example, the `minAvailable` field of the `PodDisruptionBudget` object sets the minimum number of pods that must always be available. For a node to be taken offline, you might need to increase the number of running pods to exceed the number set in the `minAvailable` field. For more information about `PodDisruptionBudgets` see the upstream Kubernetes documentation.

> **Tip**
>
> You can also use the `--disable-eviction` option with the `ocne node update` command to bypass `PodDisruptionBudget` and force pods to be deleted during the draining process. Use with caution.

Kubernetes Patch Updates

Describes updating to Kubernetes patch releases.

When an image update is detected, the image is automatically pulled, verified, and staged on each node. After the image has been staged, the node is annotated to show an update is available.

You can check whether nodes have an available update using the `ocne cluster info` command. When an OCK image is ready to install, the output of this command shows the `Update Available` field is set to `true` for a node.

> **Note**
>
> We recommend you run the `ocne cluster info` command often to check for updates. If you miss an update, and a new one becomes available, the latest one is pulled and staged and ready to use. The latest patch image is always made available on the node. If you miss a patch update, you can install the latest.

When an update is staged, use the `ocne node update` command to reboot the node to use the new image.

Installing a Kubernetes Patch Release

Update the Oracle Container Host for Kubernetes (OCK) image on Kubernetes nodes to install the latest Kubernetes patch release.

Each node in a Kubernetes cluster periodically polls the container registry to check for patch updates to the Oracle Container Host for Kubernetes (OCK) image it's running. When an update is detected, the image is automatically pulled, verified, and staged on each node, and the nodes are annotated to show an update is available.

1. Confirm an update is available for the cluster nodes.

   Use the `ocne cluster info` command to confirm the nodes are staged with an updated OCK image. Use the syntax:

   ```
   ocne cluster info
   [{-N|--nodes}] nodename, ...
   [{-s|--skip-nodes }]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster info
   ```

When an OCK image is available, staged, and ready to install, the output of this command shows the `Update Available` field to be `true` for a node.

2. Update the control plane nodes.

   Update the control plane nodes, one node at a time, with the staged OCK image.

   Use the `ocne node update` command to update each node. Use the syntax:

   ```
   ocne node update
   [{-d|--delete-emptydir-data}]
   [{-c|--disable-eviction}]
   {-N|--node} name
   [{-p|--pre-update-mode} mode]
   [{-t|--timeout} minutes]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne node update --node mynode
   ```

   Replace `mynode` with the name of the control plane node.

   > **Tip**
   >
   > After each node is updated, use the `ocne cluster info` command to check the update is complete. Node updates are asynchronous. The update is complete only when the output of this command reports an update is no longer available for a node.

3. Update the worker nodes.

   Use the `ocne node update` command to update each worker node.

Kubernetes Minor Updates

Describes updating to Kubernetes minor releases.

You can upgrade a cluster to the next Kubernetes minor version when an OCK image becomes available for that version. To do this, you use the `ocne cluster stage` command to set the target Kubernetes version.

The target Kubernetes version must be the next available minor version. For example, to upgrade from Kubernetes Release 1.29 to 1.31, first set the target Kubernetes release to 1.30 and update all the nodes, then set the target version to 1.31 and update the nodes again.

The nodes then poll the container registry for an OCK image for the target Kubernetes version. When an image is available the nodes pull and stage the image, and the nodes are annotated to show an update is available, in the same way as patch updates. Again, you then manually update each node using the `ocne node update` command.

> **Tip**
>
> If the cluster was created using a Kubernetes Cluster API provider (the Oracle Linux Virtualization Manager, or OCI provider), you can also update to a Kubernetes minor version using the Kubernetes Cluster API controllers. Using the Cluster API to upgrade provisions new compute instances or VMs using the new OCK image, instead of upgrading existing nodes, one at a time.
>
> For information on upgrading clusters provisioned with the OCI provider, see Upgrading an OCI Cluster to a Kubernetes Minor Release.
>
> For information on upgrading clusters provisioned with the Oracle Linux Virtualization Manager provider, see Upgrading an Oracle Linux Virtualization Manager Cluster to a Kubernetes Minor Release.

Upgrading to a Kubernetes Minor Release

Update the Oracle Container Host for Kubernetes (OCK) image on Kubernetes nodes to upgrade to the next Kubernetes minor release.

Upgrade a Kubernetes cluster to the next minor Kubernetes version when an Oracle Container Host for Kubernetes (OCK) image becomes available for that version. Use the `ocne cluster stage` command to set the target Kubernetes version.

1. Set the target Kubernetes version.

   Use `ocne cluster stage` command to stage the target Kubernetes version. The syntax to use is:

   ```
   ocne cluster stage
   [{-c|--config} path]
   [{-r|--os-registry} registry]
   [{-t|--transport} transport]
   {-v|--version} version
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster stage --version 1.33
   ```

2. Confirm an update is available for the cluster nodes.

   Use the `ocne cluster info` command to confirm the nodes are staged with an updated OCK image. Use the syntax:

   ```
   ocne cluster info
   [{-N|--nodes}] nodename, ...
   [{-s|--skip-nodes }]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne cluster info
```

When an OCK image is available, staged, and ready to install, the output of this command shows the `Update Available` field to be `true` for a node.

3. Update the control plane nodes.

   Update the control plane nodes, one node at a time, with the staged OCK image.

   Use the `ocne node update` command to update each node. Use the syntax:

   ```
   ocne node update
   [{-d|--delete-emptydir-data}]
   [{-c|--disable-eviction}]
   {-N|--node} name
   [{-p|--pre-update-mode} mode]
   [{-t|--timeout} minutes]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne node update --node mynode
   ```

   Replace `mynode` with the name of the control plane node.

   > **Tip**
   >
   > After each node is updated, use the `ocne cluster info` command to check the update is complete. Node updates are asynchronous. The update is complete only when the output of this command reports an update is no longer available for a node.

4. Update the worker nodes.

   Use the `ocne node update` command to update each worker node.

5. Confirm Kubernetes has been upgraded.

   Use the `kubectl get nodes` command to confirm all nodes have been upgraded and are listed with the updated Kubernetes version.

   ```
   kubectl get nodes
   ```

Cluster Backups

Learn about backing up a Kubernetes cluster using the CLI.

Adopting a back up strategy to protect a Kubernetes cluster against control plane node failures is important, especially for clusters with only one control plane node. High availability clusters with many control plane nodes also need a fallback plan if the resilience provided by the replication and fail over functionality has been exceeded.

The state for Kubernetes clusters is maintained in an `etcd` database. Access to the database is shared between all Kubernetes API Server instances. Taking regular backups of the `etcd` database is a critical part of a Kubernetes disaster recovery plan.

Typically, the backup contains sensitive data, such as Kubernetes Secret objects, so care must be taken to store the backups in a secure location.

If restoring from an `etcd` backup is part of a disaster recovery strategy, the integrity of the backup file is important. Backups must therefore be stored in a location with integrity safeguards.

> ⚠ **Caution**
>
> Only the key containers required for the Kubernetes control plane node are backed up. No application containers are backed up.

You don't need to bring down the cluster to perform a back up as part of a disaster recovery plan. Use the `ocne cluster backup` command to back up the key containers and manifests for all the control plane nodes in the cluster (the `etcd` database).

> ⚠ **Caution**
>
> The CLI doesn't provide a command to restore a cluster from an `etcd` database backup. For information on restoring a cluster using the `etcd` backup, see the upstream Kubernetes documentation.

Backing Up a Cluster

Back up the `etcd` database for a Kubernetes cluster using the `ocne cluster backup` command.

Use the `ocne cluster backup` command to back up the `etcd` database for a Kubernetes cluster. The syntax is:

```
ocne cluster backup
{-o|--out} path
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

Example 10-1 Back up the `etcd` database for a cluster

To back up the `etcd` database for a cluster to the current directory:

```
ocne cluster backup --out mybackup.db
```

Analyzing a Cluster

Perform analysis on the state of Kubernetes cluster using the `ocne cluster analyze` command.

This might be useful for debugging any issues with the Kubernetes cluster.

You can create a set of dump files to analyze using the `ocne cluster dump` command. Or you can analyze a live, running, cluster without first creating a set of dump files.

1. (Optional) Generate dump files of the Kubernetes cluster.

   Use the `ocne cluster dump` command to generate a dump file of the Kubernetes cluster. The syntax to use is:

   ```
   ocne cluster dump
   [{-c|--curated-resources}]
   [{-z|--generate-archive} path]
   [{-m|--include-configmaps}]
   [--json]
   [--managed]
   [{-n|--namespaces} namespace,...]
   [{-N|--nodes} nodename, ...]
   [{-d|--output-directory} path]
   [{-r|--skip-cluster}]
   [{-s|--skip-nodes}]
   [{-p|--skip-pod-logs}]
   [{-t|--skip-redaction}]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster dump --output-directory $HOME/dump
   ```

2. Analyze the state of the cluster.

   Use the `ocne cluster analyze` command to analyze the state of the cluster. The syntax is:

   ```
   ocne cluster analyze
   [{-d|--dump-directory} path]
   [{-s|--skip-nodes}]
   [{-p|--skip-pod-logs}]
   [{-v|--verbose}]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   To analyze a live cluster:

   ```
   ocne cluster analyze
   ```

To analyze a live cluster without including node data:

```
ocne cluster analyze --skip-nodes
```

To perform a basic analysis of cluster dump files:

```
ocne cluster analyze --dump-directory $HOME/dump/
```

To display more detailed information of cluster dump files:

```
ocne cluster analyze --dump-directory $HOME/dump/ --verbose
```

OS Console

Learn how to access a Kubernetes node's OS console using the `ocne cluster console` command.

Oracle CNE systems are administered through Kubernetes. If you need to directly access a node's OS for debugging and testing purposes, use the `ocne cluster console` command to start an administration console.

The console can be started with extra debugging tools that can be used for investigation and diagnosis purposes, by including the `--toolbox` option.

The `ocne cluster console` command can also be used with the `-- command` option to run commands on a node, without directly interacting with the shell. This might be helpful to return information about a node, without connecting directly to the console.

By default, the console session starts with the initial working directory set to root (`/`). If you need to access services that run on the node itself, for example the `ocne-update.service`, you can run the `chroot /hostroot` command, and `chroot` to the local file system of the node. Or, you can start the console session already `chroot`ed to the node file system, using the `--direct` option.

The `ocne cluster console` command is the method you use to access a node's OS in the cluster. The only reason to access a node using some other method, such as SSH, or a serial console, is when the node can't be accessed using this method.

For information on the credentials to use for SSH, see OCK Image User.

Accessing a Node's OS Console

Access a Kubernetes node's OS console using the `ocne cluster console` command.

1. Get the name of the node.

   Use the `kubectl get nodes` command to find the name of the node you want to access.

   ```
   kubectl get nodes
   ```

2. Connect to the node's console.

Use the `ocne cluster console` command to access the console of a node. The syntax is:

```
ocne cluster console
[{-d|--direct}]
{-N|--node} nodename
[{-t|--toolbox}]
[-- command]
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne cluster console --direct --node mynode
```

The preceding line of code starts a `chroot` session on the node and a terminal prompt is displayed. When you're finished using the console, enter `exit` to end the console session.

3. Pass a command for the console to run.

   The `ocne cluster console` command can be used with the `-- command` option to run commands on a node, without directly interacting with the shell. For example, to find the IP address of the node, you can run:

   ```
   ocne cluster console --direct --node ocne-control-plane-1 -- ip addr | head
   ```